InfoSift: A Novel, Mining-Based Framework for Document Classification

SHARMA CHAKRAVARTHY
The University of Texas at Arlington, Arlington
MANU AERY
Microsoft, India
ARAVIND VENKATACHALAM
Sendori, San Francisco
ADITYA TELANG
IBM Research, India

A number of approaches, including machine learning, probabilistic, and information retrieval, have been proposed for classifying/retrieving documents where mainly words from the documents are used without considering any potential structural properties of the document. These techniques do not specifically exploit: structural information that may be present in these documents or the importance of groups of terms that co-occur in different parts of the documents and their relationships. However, many documents, such as emails, web pages, and text documents have a basic structure which can be beneficially leveraged for the purposes of classification/retrieval.

This paper proposes a novel, graph-based mining framework for document classification by taking into account the structure of a document. Our approach is based on the intuition that representative – common and recurring – structures or patterns (not just words) can be extracted from a pre-classified document class and similarity with these extracted patterns can be effectively used for classifying incoming/new documents. To the best of our knowledge, this approach has not been tried for the classification of text, email or web pages (in general documents). First, we establish the applicability of this approach by identifying a suitable graph mining technique. Next, we establish relevant parameters and their derivation from the corpus characteristics. The notion of inexact graph match is critical for our approach both for extracting substructures as well as for identifying similar substructures.

Second, we extend our approach to multi-class classification which is essential for real-world applications. Ranking of substructures globally (across classes) is needed for this purpose. A TF-IDF-like formula is proposed for global ranking of substructures. Approaches proposed for the computation of the components of the ranking formula are discussed along with their computation challenges. Finally, extensive experimental analysis is carried out for three diverse document types (emails, text, and web pages) to demonstrate the generality and effectiveness of the proposed framework. We believe that we have established the efficacy of this framework for the classification of any input that exhibits some structure, and furthermore can be extended to other forms of inputs.

Keywords: Document classification, graph mining, representative substructures, inexact match, similarity of substructures, global ranking

1. INTRODUCTION

The problem of supervised 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 (or model) for each class. To determine the destination class for an unknown sample, it is compared with the descriptors of all classes and categorized to the class for which 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 larger problem of information management (as in emails, news items, web pages), classification allows us to group information belonging to different classes. This grouping wherein all relevant information that have close correspondence is categorized together presents many benefits. It allows us to retrieve similar objects easily as they are grouped together, and also enables search effectively for a particular sample.

This paper proposes and applies a novel approach based on graph mining for the problem of classification. As the domain of application, we have chosen to work with text. Text in our case encompasses general text (e.g., documents, news articles), emails and web pages. We have established the feasibility of our approach, formalized the metrics used for similarity, and validated our claims; and through extensive experiments on diverse data sets we have also established the effectiveness of our approach. In the sections that follow, we discuss issues and challenges related to text, email, and web page classification and our solutions.

1.1 Text Classification

Text classification is the problem of assigning pre-defined class labels to incoming, unclassified documents/text[1]. The class labels are assumed to be available 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 (IR) and a large body of work on classification is available. In this paper, 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.

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) [Joachims 1998], decision tree classifiers [Apte et al. 1998; Molinier 1997; Joachims 1998], k-Nearest-Neighbor algorithms (kNN) [Lam and Ho 1998; Masand et al. 1992; Yang 1994], and neural networks [Weiner et al. 1995; Ng et al. 1997]. Statistical techniques such as Linear Least Square Fit [Yang and Chute 1994], the probabilistic Bayesian classification [McCallum and Nigam 1992; Baker and McCallum 1998; Koller and Sahami 1997; Molinier 1997; Joachims 1998; Tzeras and Hartman 1993] and rule induction [Apte et al. 1994; Cohen 1995; Cohen and Singer 1996; Moulinier et al. 1996] are a few of the approaches used for classifying text. Also, the class of Term Frequency and Inverse Document Frequency (TF-IDF) classifiers [Salton 1991] 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 classes. These features are used to either build a vector that consists of the extracted terms qualified by their occurrence frequencies or probability scores for classification (in case of TF-IDF, Naive Bayes, kNN and others) or are used to train the classifier to learn those features corresponding to the classes to enable classification (as in decision trees, neural networks, SVMs and so on). The various text classification techniques will be considered in detail in the related work section. We will, for the time being, delimit our discussion to the problem of email classification, as the challenges of this domain are unique and not often found in the context of general text classification and the methods applied to solve the same.

1.2 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 (or email). 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 email communication. Most users are often overwhelmed by the sheer volume of emails 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 [Whittaker and Sidner 1996]. A tool (preferably automated) is badly needed for the management of email due to its sheer volume and the need to retrieve relevant emails

[1] We use the terms text and document interchangeably throughout the paper.

as needed later. Mis-classification[2] and putting it in the wrong place is as good as losing the email as it will be extremely difficult to locate later (even with powerful search tools). 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 are developed to address the problem of email classification, thereby automating the process with a good accuracy and efficiency. Certainly, the presence of an automated system can reduce the time to classify emails, search for a particular mail and retrieve relevant emails. 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 Challenges

We have focused on email classification (although the approach has been shown to be applicable, in general, to text classification) as 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 [Brutlag and Meek 2000]. Email classification is trickier than text classification as it is based on personal preferences (e.g., different email filing habits ranging from those who rarely classify emails to those who maintain a strict context-based hierarchical folder structure), 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 highlighted below:

i) 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 personal preference. This varies significantly from document classification where the class label associated with a document is, typically, independent of the user. This distinction needs to be taken into account by any approach proposed/used for email classification.

ii) Each users' mailbox is different and is constantly evolving. Folder contents as well as the number of folders 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.

iii) The information content of emails vary significantly, and other factors, such as the sender, group to which 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.

iv) 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.

v) Emails within a folder may not be *cohesive*; that is, the contents may be disparate and not have many common words or a theme. We characterize these folders on a spectrum of homogeneous to

[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; it is no wonder that library users are asked not to re-shelve a book.

heterogeneous. A folder may lose its homogeneity as it becomes dense making it difficult to associate appropriate central theme or representative structures with the folder.

vi) 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 identified above. Additionally, they can draw upon 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* [Boone 1998], Boone uses a machine learning technique, the k-Nearest Neighbor algorithm. Rule based classification [Cohen 1996], Naive Bayesian probability based classification [Rennie 2000], and the class of term-frequency, inverse frequency (TF-IF) classifiers widely used in information retrieval have been used for email classification [Boone 1998; Segal and Kephart 2000]. We will further elaborate on these in the related work section. Our approach, as a first of its kind, does not address all the challenges listed above. Ongoing further work is expected to address challenges not addresses in current work.

1.4 Web page

Web pages possess an inherent structure in the form of title, meta tags, anchors to other relevant pages, body, etc. which can be used to classify other unknown web pages. Web page classification is much more difficult than pure-text classification due to a large variety of noisy information embedded in web pages. Due to the different dimensionality and different representations of web pages, simply classifying them with techniques based on ordinary text documents would not be suitable. Hence, we use the structure of web page in order to derive patterns with relationships in order to classify them.

1.5 Problem Statement

Most of the techniques mentioned above rely on extracting keywords or frequency of words for classification. They ignore the importance of a group of related terms that co-occur. A classification system that extracts not only patterns of term associations but also the structural aspects from documents/emails of a class/folder and uses these holistically 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, this approach, that infers patterns from text/emails and relies on these for classification, has not been used for text, email, or web page classification .

The belief that the process of classification using supervised learning, which entails grouping related or similar entities, can benefit from the application of graph mining techniques has prompted this research. Data Mining is the process of discovering interesting, non-trivial, implicit, previously unknown, and potentially important information and patterns from data [Frawley et al. 1992]. There are many data mining techniques such as association rule mining [Agarwal et al. 1993] which discovers associations between items in a set of transactions, sequence pattern mining which mines frequent patterns related to time or other sequences, graph mining [Cook and Holder 2000] which discovers interesting patterns/subgraphs in an input graph (or forest), and frequent subgraphs (or FSG) [Kuramochi and Karypis 2001] in which a frequent subgraph is identified in a large database of graphs, to name a few.

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 paper 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 approach 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.

1.6 Our Approach

In this paper, we present a novel approach that adapts graph mining techniques for document, email, and web page classification [Aery and Chakravarthy 2005a; 2005b; Chakravarthy et al. 2010]. 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 characteristics of previously labeled documents or emails and these are used for the classification of unknown text samples or incoming emails. This paper presents a consolidation of our approach partially published in conferences.

Documents in a given class is likely to have a thematic or semantic correspondence and similarity of structures among 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 text 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 pre-extracted 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). Our first step was to demonstrate the applicability of the approach and analyze the parameters of importance. This was done for a single folder to establish the approach [Aery 2004; Aery and Chakravarthy 2005a]. Once the utility of the approach was established, it was extended to multi-folder classification so that real-world applications could benefit from this approach [Venkatachalam 2007; Chakravarthy et al. 2010].

1.7 Contributions

a) This paper proposes a novel graph-mining framework to document classification. Although mining techniques have been proposed earlier, to the best of our knowledge, graph mining has not been applied for classification.

b) This paper evaluates current graph mining approaches, their applicability to the problem at hand, and the adaptation of an appropriate approach for text and email classification. Graph mining, by itself, does not perform classification.

c) One of the primary contributions of this paper is the formulation of a framework using graph mining for the problem of document classification. This framework includes the formula for global rank of graphs that is analogous to TF-IDF based on words.

d) Finally, the last but not the least contribution of this research work is an extensive experimental evaluation of the proposed approach on multiple, diverse document types. We have analyzed the parameters to infer their effectiveness and inferred their values from the data set based on experimental results.

A number of factors that influence representative structure extraction and classification are analyzed conceptually and validated experimentally. We have adapted our approach to text, email, and web page classification and formulated a general framework for mining text.

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

The remainder of the this paper is organized as follows. *Section 2* presents the related work in the areas of text, email, and web page classification. *Section 3* gives a brief overview of graph based mining and a summary of Subdue substructure discovery system. *Section 4* presents an overview of the *InfoSift* classification system along with the discussion of parameters and their computation. In *Section 5*, we propose and discuss the formula for global ranking needed for multi-folder classification. Some implementation details, experimental results of diverse data sets and their comparison with the naive Bayesian technique are presented and analyzed in *Section 6*. Conclusions and future work are outlined in *Section 7*.

2. RELATED WORK

This section presents an overview of some of the extant approaches used 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. Various techniques proposed for classification include Support Vector Machines (SVM) [Joachims 1998], decision trees [Apte et al. 1998; Molinier 1997; Joachims 1998], k-Nearest-Neighbor (k-NN) classifiers [Lam and Ho 1998; Masand et al. 1992; Yang 1994], Linear Least Square fit technique [Yang and Chute 1994], rule induction [Apte et al. 1994; Cohen 1995; Cohen and Singer 1996; Moulinier et al. 1996], neural networks [Weiner et al. 1995; Ng et al. 1997] and Bayesian probabilistic classification [McCallum and Nigam 1992; Baker and McCallum 1998; Koller and Sahami 1997; Molinier 1997; Joachims 1998; Tzeras and Hartman 1993]. Also, the class of Term Frequency – Inverse Document Frequency (TF-IDF) classifiers [Salton 1991] from information retrieval have been applied to the problem of text classification. As is evident, different techniques that address the problem have been drawn from very diverse fields.

2.1 Text Classification Techniques

Support Vector Machines (SVM) were introduced by Vapnik [Vapnik 1982] and have been widely used. 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 [Vapnik 1998] 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 viewed 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 performance of Support Vector Machines for text classification tasks has been studied extensively and they exhibit a remarkably better performance than most other classification techniques [Yang and Liu 1999]. 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.

In *k-Nearest Neighbor (k-NN)* Classifiers [Lam and Ho 1998; Yang 1994], 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. 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 [Yang and Liu 1999]. 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 main drawback of this approach is about choosing k. Also, the presence of a large feature space can become problematic as the size of the training set increases.

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 text classification, 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 $T = t_1, t_2, t_3, ..., t_n$.

Naive Bayesian classifiers make a simplifying assumption of term independence 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. Although the assumption is strong, it does simplify the computation of term probabilities (as they can be calculated independently of each other). The performance of the classifier is good and compares well with other sophisticated techniques 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 techniques 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 Related Work on Email Classification

As we have elaborated earlier, the problem of email classification presents certain issues 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. Below, 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 assign emails to folders. Cohen [Cohen 1996] uses the *RIPPER* learning algorithm to induce "keyword spotting rules" for email classification. RIPPER is a propositional learner capable of handling large data sets [Cohen and Singer 1995]. 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) [Crawford et al. 2001] rule-based classification system learns rules based only on sender information and keywords. *Ishmail* [Heflman and Isbell 1995] is another rule-based classifier integrated with the Emacs mail program Rmail.

Although rules are easy for people to understand [Catlett 1991], 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 periodically, often involving a complete re-write 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 onerous, and need to be changed often to make them work properly.

2.2.2 *Information Retrieval Based Email Classification.* Segal and Kephart [Segal and Kephart 2000] use the TF-IDF for the classification of email in *SwiftFile* and 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. 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 [Rennie 2000] 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. [Sahami et al. 1998]. The *Re:Agent* email classifier by Boone [Boone 1998] 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 [Payne and Edwards 1997] uses the symbolic rule induction system *CN2* [Clark and Niblett 1989] 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 an informed decision during classification.

2.2.4 *Temporal Feature-Based Email Classification.* Kiritchenko et al. [Kiritchenko et al. 2004] employ temporal features in order to classify email messages into classes. Temporal features such as the day of the week, time of the day, etc. have been incorporated into traditional classification approaches. Relevant temporal features are extracted from emails and combined along with conventional content-based classification approaches in order to build a much richer information space to improve accuracy.

A set of emails is viewed as an *event sequence* $(c_1, t_1) \rightarrow (c_2, t_2) \rightarrow ... \rightarrow (c_n, t_n)$, where each event corresponds to an email and is represented as a pair (c_i, t_i) with c_i being the category of email (event type) and t_i being the timestamp of the email. The events are based on the timestamps. These temporal relations are then transformed into patterns called *temporal sequential patterns* which is an ordered sequence of event types $c_1 \rightarrow c_2 \rightarrow ... \rightarrow c_k$ along with an ordered sequence of time intervals $d_1 \rightarrow d_2 \rightarrow ... \rightarrow d_k$. An apriori-like algorithm has been developed to mine frequent sequential patterns in the input data set. These sequential patterns are used to classify incoming unknown emails.

2.3 Web Page Classification

Although it may seem that text classification techniques can be directly applied to solve the problem of web page classification, it is not 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 et al. [Attardi et al. 1999] argue that conventional text classification techniques are content driven and do 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 to 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 et al. [Schenker et al. 2003] 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 [Bunke and Shearer 2001]. 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 taken in this paper is quite different from the earlier approaches used for the general problem of document classification. Though a graph-based model has been used for classifying web pages, a conventional classification technique (e.g., kNN) is typically adapted to work with graphs used for the actual classification. Our approach uses alternate graph representations, graph mining for extracting substructures, prunes them, and ranks them within and across classes. 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.

3. OVERVIEW OF GRAPH MINING

Data in many applications have an inherent structure and reducing that data to a non-structural representation (as transactions) is likely to result in loss of information. Graph representation provides a natural format for preserving the inherent structural characteristics. If processing can be done directly on this representation it is likely to provide better results as the semantics of the applications (in the form of relationships) is taken into account during processing. Complex structural relationships can be modeled as graphs and they can accommodate cycles, multiple edges, only directional edges, and vertex and edge labels. Graphs model data using **vertices** (to characterize entities) and **edges** (that typify relationships). Unlike transaction mining, graph mining is used to mine structural data such as DNA sequences, electrical circuits, chemical compounds, social networks, schemes (such as money laundering and fraud) that have associations and relationships of transactions. A graph representation comes across as a natural choice for representing relationships as the data visualization process is relatively simple as compared to a transactional representation. Data representation in the form of a graph preserves the structural information of the data which may otherwise be lost if it is translated into other representations.

An email message is inherently made up of a structure that can be used for its representation and can be exploited for its classification. The structural relationships between the *headers*, *subject* and *body* of an email can be represented as a graph. Although we do not use it, presence or absence of attachments and the number of attachments can also be used. This approach of considering the structure of a document is unique from other forms of document classification that assume the document as a set (or bag) of words having no particular structure. This is also true of other documents such as web pages and text documents that have a structure in the form of title, section headings, HTML tag elements, meta tags, anchors to other pages and document body. By making use of the structural information they can be made amenable to graph mining.

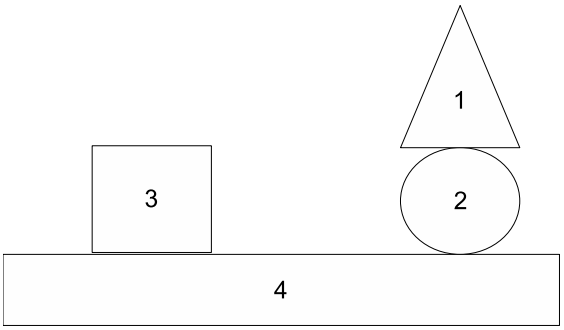

Figure 1. High-level view of shapes

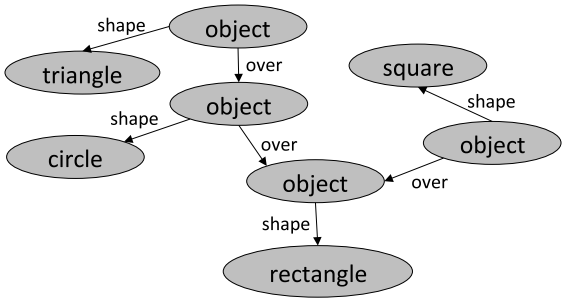

Figure 2. Graph representation of shapes example

```
v 1 object
v 2 object
v 3 object
v 4 object
v 5 triangle
v 6 circle
v 7 square
v 8 rectangle

u 1 2 over
u 2 3 over
u 2 4 over
u 3 4 over
u 1 5 shape
u 2 6 shape
u 3 7 shape
u 4 8 shape
```

Figure 3. Subdue Input for shapes example

```
Best 3 substructures:

(1) Substructure: value = 0.96959, pos instances = 1, neg instances = 0
 Graph(2v,1e):
  v 1 object
  v 2 object
  u 1 2 over

(2) Substructure: value = 0.953003, pos instances = 1, neg instances = 0
 Graph(2v,1e):
  v 1 object
  v 2 square
  u 1 2 shape

(3) Substructure: value = 0.953003, pos instances = 1, neg instances = 0
 Graph(2v,1e):
  v 1 object
  v 2 rectangle
  u 1 2 shape
```

Figure 4. Subdue Output for shapes example

3.1 Overview of Subdue

Subdue [Cook and Holder 1994; Ketkar et al. 2005], earliest work on graph mining, uses an information-theoretic model for determining the best substructure given a forest of unconstrained graphs. The Subdue *discovery algorithm* discovers repetitive patterns and interesting substructures in graph representations of input data. A substructure is a connected subgraph within the graph representation. Within the representation, entities and objects are mapped to vertices and the relationship between these objects is represented as an edge between the corresponding pair of vertices. An instance of a substructure in an input graph is a set of vertices and edges from the input graph that match the graphical representation of the substructure. The input to Subdue is a forest of graphs and the output is a set of substructures that are ranked based on their ability to compress the input graph using the *Minimum Description Length* [Cook and Holder 1994] (MDL) principle. The compression method used in Subdue is elaborated later.

The input is in the form of a table consisting of a list of unique vertices in the graph and corresponding edges between them. The output is list of representative substructures discovered in the input graph where each is qualified by its size and occurrence frequency in the input graph. Consider the example in Figure 1. It is a high-level view of shapes resting on a table. The graphical representation of these shapes is shown in Figure 2.

The input for Subdue (for this particular example) is shown in Figure 3. This input is given as a file consisting of the list of vertices and the edges between the vertices. Subdue generates the best substructures that compress the input graph the most and lists out the top n substructures. The output given by subdue for the example in Figure 3 is displayed in Figure 4. The following section briefly explains the Subdue's *substructure discovery process*.

3.1.1 *Substructure Discovery in Subdue.* The substructure discovery in Subdue is done by using a constrained search (using the notion of a *beam*) and progresses in an iterative manner stating with substructures of size 1 (1-edge substructures) and expanding to successively larger substructures. A list consisting of the best substructures to be expanded (specified by the *beam*) in the next iteration is maintained. The input graph is compressed by replacing the instances of substructures by a single node and the MDL value is computed. Substructures are ordered by the MDL value in descending order. This process continues until the number of passes specified by the user is reached or it meets one of the several halting conditions, such as the total number of substructures needed, provided by the user.

The occurrences of substructures that have an exact match are unlikely to occur in many domains. Substructure instances that are not exactly same but are similar can also be discovered by Subdue. Subdue is capable of discovering both exact and inexact (isomorphic) substructures in the input graph. Subdue employs a *branch and bound algorithm* that runs in polynomial time for inexact graph match and discovers graphs that differ by a *threshold* given by the user. This discovery process finds repetitive and hence interesting substructures or patterns and identifies best substructures that compress the graph most as measured by the MDL value.

3.1.2 *Compression and Evaluation of Substructures.* There are two schemes that Subdue uses for evaluating the candidate substructures in order to determine the best substructures. They are:

Compression based on MDL principle. The MDL principle states that the best theory to describe a set of data is one that minimizes the description length of the entire data set. The description length corresponds to the number of bits required to encode the input. This theory was described by Rinssanen [Rissanen 1989] and has been used in various applications such as decision tree induction, image processing, and others. Subdue employs this principle for substructure discovery where the best substructure is the one that minimizes the description length of the original input graph. According to the principle, the description length of the input graph is given as

$$DL(S) + DL(G|S) \tag{1}$$

where,

S is the discovered substructure

G is the input graph

$DL(S)$ is the number of bits required to encode the substructure

$DL(G|S)$ is the number of bits required to encode the input graph G after it has been compressed by the substructure S

The final value of the MDL is defined as

$$MDL = \frac{DL(G)}{DL(S) + DL(G|S} \tag{2}$$

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

$$Compression = \frac{1}{MDL} \tag{3}$$

Compression based on size of the graph. 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

$$\frac{Size(G)}{(Size(S) + Size(G|S))} \tag{4}$$

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.

3.1.3 *Inexact Graph Discovery.* Inexact graph discovery in Subdue aids in grouping similar substructures as a single substructure for both identification and representation. The algorithm developed by Bunke and Allerman [Bunke and Allerman 1983] is used for inexact graph discovery where a cost is assigned for each dissimilarity. The distortion between two substructures might be a variation in the edge or in the vertex descriptions like an addition, deletion or substitution of vertices or edges. Two substructures are considered to be isomorphic as long as the cost difference in generating both the substructures to be identical falls within the range the user considers acceptable. Even finding similar substructures is an NP complete problem and requires an exponential algorithm. Subdue uses a branch and bound algorithm that is executed in polynomial time by considering reduced number of mappings. The *threshold* parameter is used in order to control the number of differences between two substructures. Grouping similar substructures as the same substructure forms an integral part in classification of documents which we will elaborate in later sections.

The concept of inexact graph match is one of the most important aspects of our approach. It allows for substructures that vary slightly in their vertex or edge label descriptions to be chosen as instances of a single substructure. The amount of variation permissible is determined by the *threshold* parameter provided by the user. It specifies the bound on the difference that is allowed between instances of a substructure. Subdue assigns all transformations (insertion, deletion of an edge or vertex and so on) between instances an uniform cost of 1. For a given substructure instance *inst*, to be classified as an instance of another substructure *sub*, the following condition needs to be satisfied:

$$matchcost(sub, inst) \leq size(inst) * threshold \tag{5}$$

In other words, the total transformation cost needs to be less than the number determined by the particular value of threshold and substructure size.

If the size of substructures is large, then even with a small value of $threshold$, there can be a large variation in the edge and vertex labels of the two instances being considered. The default value for $threshold$ is 0.0, which means that the graphs have to match exactly. A very large value of $threshold$ may not be meaningful as it will match two dissimilar graphs. The exact value of $threshold$ has to be determined from the input being processed; knowledge of folder characteristics is crucial for determining the threshold.

3.2 Parameters for Subdue substructure discovery

There are a number of parameters that Subdue accepts from the user in order to control the flow of the substructure discovery process. The input to Subdue is a file containing the list of vertices and corresponding edges as shown in Figure 3. Some of the parameters used by Subdue are briefly described below:

BEAM. This parameter specifies the number of top (or best) substructures that are retained for expansion in each iteration of the discovery algorithm. The default value of the *beam* is 4.

ITERATIONS. This parameter specifies the number of passes (different from the number of iterations in each pass as the size of the substructure is increased) to be made over the input graph. The best substructure from the previous pass is used to compress the graph for the next pass. The default is no compression or one pass. Note that within each pass, the best substructure is identified by generating substructures of increasing sizes using iterations and computing their compression quality.

LIMIT. specifies the number of different substructures to be considered in each iteration. The default value is *(number of vertices + number of edges)/2*. This is different from the beam.

NSUBS. is used to specify the number of substructures to be returned from the total number of substructures that Subdue discovers.

OUTPUT. This parameter controls the screen output of Subdue. The various values are

i) Print the best substructure found in each iteration.
ii) Prints the best 'n' substructures, where n is the number specified in the nsubs parameter.
iii) Print the best 'n' substructures, as well as the substructure instances.
iv) Print the best 'n' substructures along with their instances and intermediate substructures as they are discovered.
v) Same as above, prints also each substructure considered.

OVERLAP. Specifying this parameter to Subdue allows the algorithm to consider overlap in the instances of the substructures. Instances of substructures are said to overlap if they have a common substructure in them. 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.

PRUNE. If this parameter is specified, then the child substructures whose MDL value is lesser than their parent substructures are ignored. Since the evaluation heuristics are not monotonic, pruning may cause Subdue to miss some good substructures, however, it will improve the running time. The default is no pruning. Also, acts as a halting condition when no child substructures are left in the child list on pruning.

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.

THRESHOLD. This is the 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 should match exactly. Currently, Subdue supports threshold values up to 0.3.

As our focus is on the validation of the proposed approach, the *InfoSift* System uses Subdue as a block box and tweaks available parameters beneficially. Only where absolutely needed, individual modules of Subdue are used for specific computations to gain efficiency.

4. OVERVIEW OF *INFOSIFT*: A GRAPH-BASED DOCUMENT CLASSIFIER

The *InfoSift* system, developed as part of this work, aims at automating the process of document classification as much as possible by using graph mining techniques to classify documents into pre-determined 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 Figure 5. While this section describes the overall architecture and the effectiveness of this approach for classification, multi-folder classification are further discussed in the next section.

4.1 System Overview

The document classification process is divided into two phases: Training and Classification. The classifier is first trained on a set of data where substructures are generated, pruned, and ranked. Then the classifier uses these substructures to classify incoming unknown documents. A brief description of each step followed during training and classification phases as shown in Figure 5 is given below.

(1) **Pre-Processing**: The documents in the training set contain stop words and different forms of the same word which need to be eliminated in order to generate interesting substructures. *Stop words* (such as 'as', 'is', 'to', 'from', etc.) *Elimination, Stemming, and Feature selection* are done before various characteristics of the class are calculated in order to derive the parameters for substructure discovery. Some examples of the class characteristics are *average size of documents in a class*, *number of unique words*, etc. Details of stop word elimination and Stemming are discussed in Section 4.2.

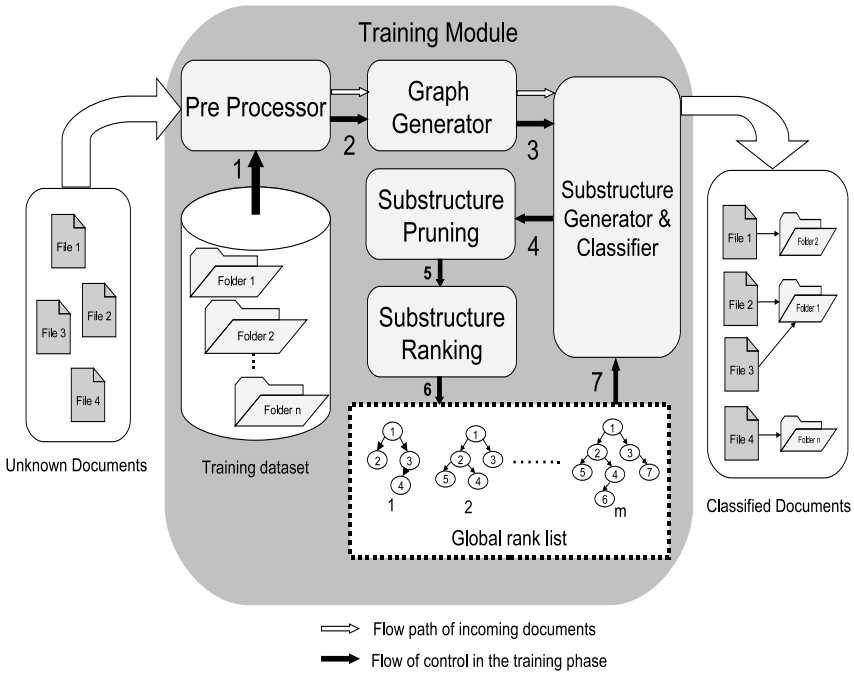

Figure 5. *InfoSift*: System Overview

(2) **Graph Representation**: Once the training documents have been pruned and the respective folder characteristics have been computed, the documents are represented in the form of graphs using the Graph Generator module. Text, emails, and web pages have a definite inherent structure associated with them which is used to generate graphs. We have proposed different canonical graph representations in order to generate substructures with better structural representation. The details about the different schemes are elaborated in Section 4.3.

(3) **Substructure Extraction**: Subdue (described in Section 3) is used to extract interesting and representative substructures from the training data set. The parameters used for Subdue is based on the folder characteristics which are computed during the process of representing the input in the form of graphs. One of the salient parameters is the *threshold* which allows grouping of similar substructures to be considered as the same substructure.

(4) **Substructure Pruning**: The output of the discovery process generates a large number of substructures; however, only a percentage of them contribute towards the classification process since the variation observed amongst majority of these substructures is only minimal. Furthermore, the cost of retaining and processing all the generated substructures is high and certainly not desirable in an already expensive processing technique (graph mining). Due to these reasons, the substructures are pruned before they are ranked and employed for classification. The aim of pruning is to identify only those substructures that would help in discriminating the unknown documents during classification. Hence, the output substructures are pruned based upon a range of conditions (elaborated in Section 5) and only those which *cover* a significant portion of the class are retained.

(5) **Substructure Merging and Ranking**: Once all the substructures from different classes have been generated and pruned, they are merged into a single list in order to be ranked. Some representative substructures occur more frequently or measure well in terms of size; hence, these can be considered to be more important than the others. It is therefore important to discriminate the substructures from the view point of classification. Certainly, there is a difference in a match with a highly ranked substructure versus a lower one. The ranking of substructures globally across all the training classes eliminates the problem of selecting the order of folders for searching. More details about ranking are

discussed in Section 5.

(6) **Processing incoming unknown Documents**: Pre-processing (similar to the one applied to the training samples, such as stop word removal, stemming, etc.) is applied to the unknown sample to be classified to bring it into a canonical representation. The canonical representation of this document is then converted into a graph for classification.

(7) **Classification**: The test document, augmented with the graph representing the substructure in the ranked order, is fed to the classifier to check for any occurrences of the substructure in the test document. The test document is grouped to the same class as that of the highest ranking substructure that occurs in it.

4.2 Pre-Processing

A document usually contains a number of unnecessary words that can adversely affect the classification. Moreover, using the entire document, with all the words, would be overwhelming. For example, documents consists of articles, conjunctions and other common words that occur frequently across all the documents; as they do not aid in classification of the document they can be pruned without affecting the outcome. Even inflected and derived words, such as 'eat','eating','ate', etc., could be reduced to their stems in order to preserve the semantics and yet reduce the number of unique words in the document. 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 mining approach.

Several techniques have been used for pre-processing documents in order to prune the size of input to retain only interesting words. The main goal of pre-processing in *InfoSift* is to retain frequent substructures that occur across documents. In order to achieve this, all the words that comprise the substructures have to be retained in the document as well. The terms have to occur frequently across all documents instead of a single one. This notion of retaining the frequent words across the documents takes care of the disparity of some documents being longer than others. Therefore, prior to representing the documents as graphs, the documents are pre-processed.

4.2.1 *Stop Word Elimination.* Stop words, such as conjunctions, articles, and even common words that occur frequently across all documents, are eliminated. Some of the more frequently used stop words for English include 'a','of','the', 'I', 'it','you', and 'and'. These are generally regarded as 'functional words' which do not carry meaning (are not as important for communication). The assumption is that the meaning can be conveyed more clearly, or interpreted more easily, by ignoring these functional words. Stop word elimination is performed by many search engines in order to assist users with queries to provide better results by avoiding searching for functional words.

Consider the document shown in Figure 6. The conjunctions or articles in the document do not assist in generating interesting substructures or in classification. Consequently, the words considered for representing a document are those which occur frequently, preferably across all the documents in a given class and not merely in a single document. Assuming the set of words in that document sample is as shown in Figure 7, the frequent set considered for further processing after stop word elimination is displayed in Figure 8.

4.2.2 *Stemming.* Stemming is the process of reducing inflected words to their roots/base/stem. This process reduces the number of unique words through out the documents and also aids in classification. For example, the words 'seeing', 'see', 'seen' are all reduced to the same word 'see'. Words ending with 'ed', 'ing', 'ly', which are used to represent the tenses of the verb or adjectives in English grammar are stripped to their root. A problem with Stemming might be *homograph disambiguation* which means a single word can have more than one meaning. For example, the word 'saw', which would be reduced to its root 'see', like in the previous example, can also mean the tool used in carpentry to cut wood. Since the advantages of stemming outweighs its limitations, it is applied as a part of our preprocessing.

4.2.3 *Feature Selection.* Feature Selection [Guyon and Elisseeff 2003] or feature reduction is a technique commonly used in machine learning approaches for selecting a subset of relevant features in order to build the learning model. This process removes the most irrelevant and redundant features from data

CCC AUTHORIZES ADDITIONAL AID

The Commodity Credit Corporation CCC, has authorized an additional 8.0 mln dlrs in credit guarantees for sales of vegetable protein meals to Hungary for fiscal year 1987, the U.S. Agriculture Department said. The additional guarantees increase the vegetable protein meal credit line to 16.0 mln dlrs and increases the cumulative fiscal year 1987 program for agricultural products to 23.0 mln dllrs

Figure 6. Document Sample

Words
This
is
Guarantees
Commodity
Increasing
Credit
Corporation
Dollars
Agriculture
of
Protein
Meal
Vegetable
to
said
Million
. . .

Figure 7. Words in Document Sample

and also helps improve the performance of learning models by enhancing the generalization capability and ameliorates the learning process. In our system, words, after stop word elimination and Stemming, are ranked based on their occurrence frequency across the documents in a class and only those words whose frequencies account for more than *f*% of the sum of all frequencies are retained. Occurrence of unique words across different folders are counted while multiple occurrences of the word in the same document are not considered. Words that are a part of this frequent set are considered for the generation of graphs. The intuition behind this being that lower frequency words may not contribute towards classification. The parameter *f* is tuned to observe its effect and identify any possible dependency on the effect of classification. This ensures the words chosen are frequent not only in a single document, but across a substantial number of documents in class.

Consider the sample document, belonging to a class, in Figure 6. To construct the graph corresponding to this document, the set of frequent terms across all the documents is considered. Assuming the set of frequent terms is as shown in Figure 9 and the feature selection parameter, *f*, as 90, the top words that correspond to 90% of the sum of frequencies of all words in the documents are retained. In our example, the sum of frequencies comes to 101 and 90% of 101 is approximately 90. Only the top *n* words, whose sum of frequencies add up to 90, are considered for graph generation. The words in the frequent set after feature selection are illustrated in Figure 10.

4.3 Graph Representations

Graph representations are chosen based on the domain knowledge so that they represent the domain under consideration appropriately. For example, information about the structure of an email message or the

Words
Guarantees
Commodity
Increasing
Credit
Corporation
Dollars
Agriculture
Protein
Meal
Vegetable
said
Million

Figure 8. Words after Stop Word Elimination

Words	Occurrence Frequency
Guarantees	22
Commodity	17
Credit	14
Corporation	12
Dollars	11
Agriculture	7
Protein	7
Meal	5
Vegetable	2
Increase	1
Fiscal	1
Million	1
Year	1

Figure 9. Frequent Set of words

Words	Occurrence Frequency
Guarantees	22
Commodity	17
Credit	14
Corporation	12
Dollars	11
Agriculture	7
Protein	7

Figure 10. Frequent Set after Feature Selection

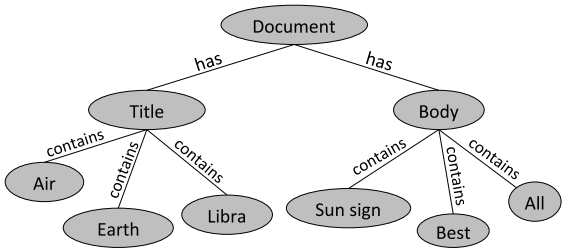

Figure 11. Tree Representation of a Text Document

structural layout of a web page would help in representing each document as a graph. We have proposed two graph representations that can be used across different domains such as text, emails, and web pages. The canonical representation shown in Figure 11 is a tree representation. The graph starts with the type of document as the root and then branches out based on the domain. In this example, the document is a text and hence the root is attached to two other vertices, title and body. All the words in the title and body of the document are attached to the title and body vertex respectively with the edge label as *contains*. The representation of an email message under this representation is shown in Figure 12. This scheme considers most of the information in an email message (attachments, cc, bcc can also be included) with each word in the email connected to the central root vertex[3].

Figure 13 illustrates an alternative graph representation, termed *star representation*, developed to be used across different domains. It consists of a central anchor or root vertex. The chosen words from the document form the the remaining vertices, along with the edges that connect them to the central root vertex with the edge *contains*. The example shown in Figure 13 is for a document. A star representation of an email would contain 'Email' as the central vertex and corresponding labels directing to the other

[3]Most emails are replies to other emails in the thread and usually contains a large body of previous emails. They are not included in the representation for obvious reasons.

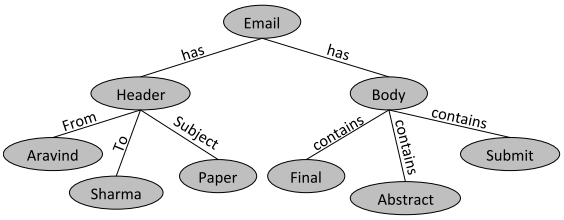

Figure 12. Tree Representation of an Email

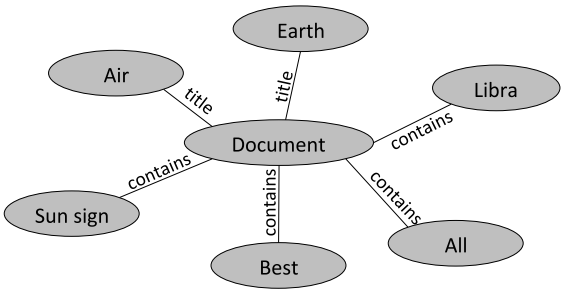

Figure 13. Star Representation of a Text Document

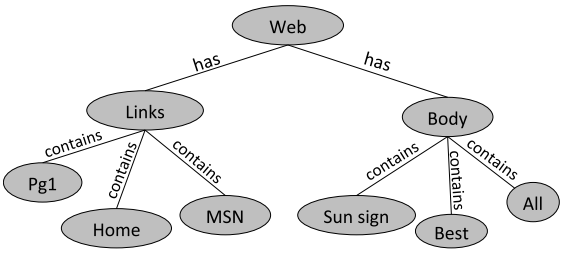

Figure 14. Tree Representation of a Web Document

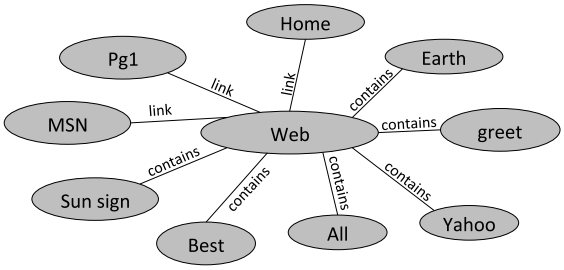

Figure 15. Star Representation of a Web Document

vertices constructed from the message. The ability to label edges makes this simple representation quite effective if the labels corresponds to the various components of a document, email or web page. This representation also comes close to the bag of words used in mining without using structural information.

Figure 14 shows the representation of a web page in the form of a graph that takes into account the information represented by the title of the page, hyper links that point to other pages and the information represented in the page. Hyper links have also been represented in the graph since they point to information sources relevant to the current page. The star representation of a Web document is shown in Figure 15.

Once the documents have been represented as graphs (a forest of graphs), they can be mined for finding representative substructures. The input file to the graph miner consists of vertex and edge entries corresponding to the graphs. Each vertex entry associates a unique vertex id with every vertex label. Each entry corresponding to an edge is represented as an undirected edge between a pair of vertices and the corresponding edge label. The input file to the Subdue system corresponding to the representation in Figure 13 is shown in Figure 16.

The discovery process is driven by certain parameters that are determined as part of the pre-processing and the graph generation phase. The value of parameters used warranted a detailed study and are explained below.

4.4 Computation of Folder Characteristics

Our approach relies on the identification and extraction of representative substructures from a given class of documents and use them for classification of unknown documents. In order to achieve this, we have to choose a number of input parameters for the Subdue algorithm that determine the number and type of substructures identified and retained during substructure discovery. The training set of classes is used as a source in order to derive these parameters (for the reasons outlined in Section 1.3). Certain characteristics of the class need to be taken into account in order to determine representative substructures that best characterize a particular class. These parameters must be tunable and effective for diverse document and class characteristics. Not all classes exhibit similar properties; certain classes may be dense as compared to others and others may have larger document content providing extensive amounts of information for

```
V 1 Document
V 2 Air
V 3 Earth
V 4 Sun sign
V 5 Best
V 6 All
V 7 Libra

U 1 2 Title
U 1 3 Title
U 1 4 contains
U 1 5 contains
U 1 6 contains
U 1 7 contains
```

Figure 16. Input to Subdue of Sample Document

Class Size	*nsubs* Formula
Small	1.25 X C_s + 0.50 X avg_s
Medium	0.90 X C_s + 0.50 X avg_s
Large	150 + 0.50 X avg_s

Figure 17. Formulas for calculating nsubs

training the classifier. For instance, in the email domain, due to constant addition, deletion, and movement of emails, the folder contents keep changing rapidly. Hence, 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 substructures generated are likely to better reflect the contents of the class. Some of these characteristics, which we believe are critical, are elaborated below. Note that these formulas have been arrived at after extensive experimental evaluation to understand their effect on various aspects of classification. See [Aery 2004; Aery and Chakravarthy 2005a] for more details.

4.4.1 *Average Document Size, Discovery and Classification Threshold.* In the text domain, when matching words it is impractical to find instances that match exactly. For the purpose of classification, flexibility in matching instances of substructures is important. This latitude in matching similar instances should be applicable while building the descriptor for the document as well as while comparing an unknown sample with the class descriptor. This matching of similar instances is carried by the inexact graph match in Subdue. As discussed in Section 3.1.3, the threshold parameter determines the amount of inexactness between two instances. This is accomplished by determining the number of vertices and edges that vary among the instances of the same substructure. The actual number is determined by Equation 6.

$$(num\,of\,vertices + num\,of\,edges) \times threshold \tag{6}$$

Since the threshold need to be small if the graph size is more and vice versa, it need to be computed based on the size of the representative substructures. At the same time, the maximum number of changes need to be capped appropriately for larger structures as well as smaller substructures without which dissimilar substructures will be deemed as similar. A small value of *threshold* allows a significant amount of inexactness while comparing substructure instances of documents that contain a large number of words. It is because even with a small value, the value computed by Equation 7 would allow reasonable number of variations. However, for documents with relatively smaller content and hence fewer vertices in the input graph representation, a larger value of threshold is required. Employing the size of the documents in a class, we can determine the amount of inexactness to allow for a graph match. If the amount of inexactness to be allowed in terms of the number of edge/vertex label variations is '*i*', then value of threshold is computed as in Equation 7

$$threshold = \frac{i}{avg_s} \tag{7}$$

where, avg_s is the average size of the documents in the class. The above formula derives a good value

for threshold taking into account the size of the documents in the class. For smaller documents, the value of the threshold will be larger compared to that of larger sized documents. In any case, the maximum number of variations is capped at 4 (larger values only lead to different substructures being considered as similar substructures which in turn lead to increase in misclassification). Here, we have interpreted average document size as a parameter that affects pattern discovery and is used it to compute the value of threshold that allows for a reasonable amount of variation and at the same time, preserves the similarity between instances. The value of threshold is used during substructure discovery process and further during classification.

4.4.2 *Number of Substructures.* The number of substructures returned by Subdue 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 need also be derived from the class characteristics. If there are a large number of documents in a class, there probably will be a large number of substructure instances as well. But all of these substructures do not aid in classification. 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 is given in the Equation 8.

$$nsubs = w_1 \times C_s + w_2 \times avg_s,\ w_1 > w_2 \tag{8}$$

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

The formula for *nsubs* is built on two class characteristics: 1) Size of the class and 2) Average document size in the class. As evident, the size of the class has got a greater impact in deriving the value for *nsubs*. Classes have been discriminated into small (less than 60 documents), medium (61 to 200) and large (Greater than 200 documents) based on the number of documents contained within them. The value of w_1 is based on the class size. The formulas for calculating *nsubs* based upon the class size in shown in Figure 17.

Subdue generates and retains substructures based on their ability to compress the original graph. Hence, for a smaller class, large substructures, despite their low frequencies, are chosen as best substructures because abstracting even their few instances results in greater compression. To make sure that smaller substructures with higher frequencies are also considered, a larger value of *nsubs* is required. Therefore, taking into account the need for a large *nsubs* with a small class size and scaling it to increase in average document size, w_1 has been assigned a value of 1.25 and 0.50 for w_2 for small classes. These values have been determined based on experimental observations for the *InfoSift* framework [Aery 2004]. The weight w_2 is fixed at 0.50 for all average document sizes.

However, for medium classes, it is likely that repetitive substructures, rather than isolated instances of long substructures will be reported as the best substructures. Thus, 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 a value of 0.90 for w_1. Classes with more than 200 documents are considered to be large and an increase in class size thereafter will serve to increase substructures instances rather than the number of substructures themselves. Consequently, the term corresponding to the class size has been capped at 150 to include the top most frequently occurring substructures.

4.4.3 *Beam.* As explained in Section 3.2, *beam* determines the number of best substructures retained at the end of each iteration of the discovery algorithm. *Beam* ensures that interesting substructures discovered during each set of iterations are available for further consideration. The *beam* value is chosen in proportion to the class size. Large sized classes typically contain many patterns owing to the presence of a large number of documents. A low value of *beam* results in loss of some interesting substructures while a larger value of *beam* only increases the computation and processing time. Hence, the *beam* values have to be chosen based on the class size to ensure no interesting substructures are missed. Experiments employing different beam values on different class sizes were performed. Beam value of 4 returned good

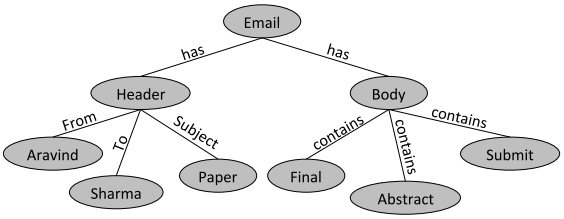

Figure 18. Tree Representation of Sample Email

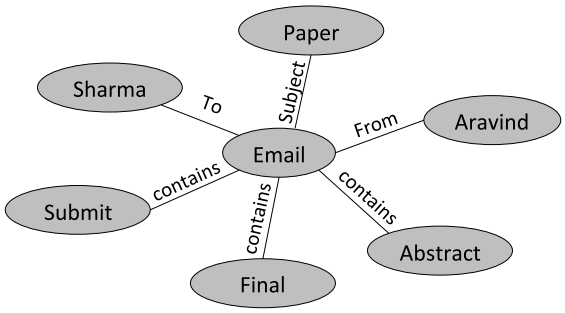

Figure 19. Star Representation of Sample Email

results and have been used for experiments. Larger value of beams only leads to increased computation time and resources during substructure discovery and pruning of unwanted substructures while a smaller value of beam did not include many interesting substructures.

4.4.4 *Minimum Size.* The representative substructures that are chosen should provide enough information for differentiation for classification. Substructures that are common across all the folders/emails provide no differentiating capability. For instance, a substructure that consists of information regarding only the headers of emails, like sender and addressed, will not help in classification as emails with same information will be hard to differentiate from each other. This is not an acute problem for single folder classification (where the email is classified to one folder or not). However, for multiple folder classification, this problem needs to be avoided. It becomes vital to have further information for enabling successful multi-folder classification. The representative substructures chosen should provide enough information for discrimination amongst folders. The size should be constrained above a minimum size to retain substructures that contain information more than just a common 'core'.

From the graph representation of Figure 18, it can be inferred that the smallest sized substructure contain at least four vertices (Email, Header and any two among 'To', 'From','Cc'). 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 at 4. Using a lower minimum size result in a lot of misclassification of the test documents. This constraint needs to be determined from the graph representation scheme employed.

For the star representation shown in Figure 19, the minimum size of the substructure should be 3. This ensures that the substructures that are picked have sizes greater than the size of substructures that are most likely to be common across many document folders, and hence capable of discriminating between the same.

Having provided a detailed description of the parameters that affect substructure discovery, their importance for our classification of incoming test documents, as well as how these parameters are derived based on the data set characteristics, we now elaborate the post discovery processing steps for multi-folder classification in Section 5 and brief implementation along with experimental analysis in Section 6.

5. MULTI-FOLDER CLASSIFICATION

First, we tested our premise on single folder classification. This entailed extensive experiments on all types of documents (email, web pages and text) to infer parameters needed for using Subdue. We also compared the accuracy of single folder classification using graph mining with naive Bayes approach with positive results [Aery 2004; Aery and Chakravarthy 2005a]. This section discusses multi-folder classification and elaborates on the processing steps after the representative substructures have been discovered. The generation of representative substructures is explained in Section 3 using the Subdue's substructure discovery process. The goal of the classification system is to classify an unknown document into a folder that best matches the incoming unknown document characteristics. In order to achieve this, representative substructures are generated based upon the input folders' (training set) characteristics, ranked globally,

and are used to classify the incoming test documents to determine the best matching folder for that input.

5.1 Substructure Pruning

The substructure discovery process generates the top *nsubs* substructures from the training set. Retaining and processing all of these substructures is not necessary since a majority of these substructures are small variants of each other and the classification is not affected due to the use of inexact match. Retaining several substructures that have the same frequency and size but vary only sightly in terms of content will not aid in distinguishing the incoming document and will only contribute towards increasing the processing time. Therefore, pruning is necessary in order to retain only those substructures that truly represent the class and are discriminating during compression in each iteration. Pruning of substructures is required in the following two cases.

Substructures with same frequency, size and MDL value. Substructures differing in either frequency, size, or MDL value are retained to ensure uniqueness. For example, two substructures, each having ten vertices and different occurrence frequency, are not similar since the same substructure is not reported twice with different occurrence frequency. But two substructures, each with ten vertices and same occurrence frequency with minimal difference, such as different vertex or edge label, add to the overhead, but not for classification, and hence are pruned. Therefore, each substructure in the representative set refers to an unique pattern that follows from the documents of the class under construction.

Substructures with low frequency in large classes. On account of using compression as a heuristic, the discovery algorithm also identifies certain large substructures that do not occur frequently. It is due to the fact that replacing these huge substructures greatly compresses the original input graph (or forest). Hence, these substructures are returned by Subdue as part of the best substructure list even though they do not occur frequently. These substructures do not significantly add to the substructure set as they do not cover substantial portion of the class contents. Therefore, substructures with very low frequencies as compared to the class size are discarded from consideration. The representative substructures generated from different categories (folders) are generated and pruned to retain unique substructures for ranking.

5.2 Substructure Ranking

The representative set of substructures are generated and pruned separately for each folder. Thus, all the folders in the training set have a list of pruned representative substructures. In order to classify the incoming test document to the best category exhibiting similar characteristics, the test document has to be matched against the representative substructures in each category and the best match has to be computed. Ranking of the representative substructures is important in order to position the representative substructures in an ordinal scale in relation to each other. The single folder *InfoSift* framework ranked the substructures in each category in relation to the other substructures in the same category only.

For multi-folder classification, the pruned representative substructure list of each folder is collected and appended into a single list to rank them globally. This ranking scheme is critical for the accuracy of multi-folder classification. A rank is associated to each of the substructures based on the formula in 9:

$$Rank(RS) = \left[\frac{FRS(f, RS)}{FRS(A, RS)} \times \frac{1}{IFF(RS)^2}\right] \times \frac{S_{RS}}{Max_{RS}} \tag{9}$$

where,

RS is the Representative Substrucutre
$FRS(f, RS)$ is Frequency Term of RS across folder f
$FRS(A, RS)$ is the Frequency Term of RS across all the folders in Training set represented as A
$IFF(RS)$ is the Inverse Folder Frequency of RS
S_{RS} is the Size of RS
Max_{RS} is the Size of the largest RS in the global list

Equation 9 computes the rank of the representative substructure *RS* globally across all the folders in the training set given as the input to the learning model. The rank comprises of two characteristics of

the substructure in question: i) its *occurrence frequency* in both – the folder it was extracted from and the training set, and ii) its size. The intuition and details of these characteristics and the method of computation for the same are discussed in the following subsections.

5.2.1 *Frequency of Representative Substructure.* Representative substructures represent a group of words that co-occur (along with occurrence structure) throughout the document class. It is relevant to calculate their occurrence frequency in order to rank them in comparison with each other. The frequency term is shown in the enclosed square brackets in Formula 9. It is based on the principle that the weight assigned to the representative substructure is proportional to its frequency in the folder it was extracted from and inversely proportional to its frequency across other folders. The frequency term comprises of three elements, thus, elaborating the importance of the representative substructure based on its occurrence frequency. This formula is somewhat based on the TF-IDF principle, but takes substructure frequency rather than word frequency. The three elements are elaborated below.

5.2.2 *Frequency Term of Representative Substructure in folder* f. $FRS(f, RS)$ (Frequency of Representative Substructure in folder f) term captures the importance of the group of words that relate together in the representative substructure RS. The value of $FRS(f, RS)$ is in turn computed by equation given in 10:

$$FRS(f, RS) = \frac{freq(RS, f_{RS})}{\sum_{i=0}^{n} freq(RS_i, f_{RS})} \tag{10}$$

where
f_{RS} is the folder from which RS is generated
$freq(RS, f_{RS})$ is the frequency occurrence of RS in folder f_{RS}
$freq(RS_i, f_{RS})$ is the frequency occurrence of ith RS in folder f_{RS}
n is the total number of substructures within f_{RS}

The frequency of RS in f is given by Subdue in its output of best substructures. For example, in Figure 4, the *positive instance* for each representative substructure denotes its frequency of occurrence across the folder. The frequency of all the representative substructures across folder f can be computed by summing up the positive instances of all the representative substructures that have been extracted in folder f. This term evaluates to a *high value* when the representative substructure occurs more frequently across the document class it was generated from.

5.2.3 *Frequency Term of Representative Substructure in all folders* A. This term represents the commonality of the representative substructure by computing the occurrence of RS through out the training set. The importance of this representative substructure RS is inversely proportional to its occurrence in other document classes since it does not aid in classification of test documents. This term can be computed by the equation 11:

$$FRS(A, RS) = \frac{\left[\sum_{j=0}^{m} freq(RS, f_j)\right] - freq(RS, f_{RS})}{\sum_{j=0}^{m}\sum_{i=0}^{n} freq(RS_i, f_j)} \tag{11}$$

where
f_{RS} is the folder in which RS is generated from
$freq(RS, f_j)$ is the frequency occurrence of RS in folder f_j
$freq(RS_i, f_j)$ is the frequency of RS_i in folder f_j
m is the total number of folders in the training set

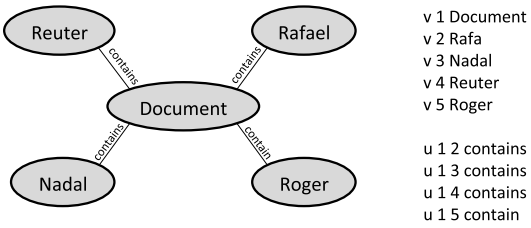

Figure 20. Sample Graph g1

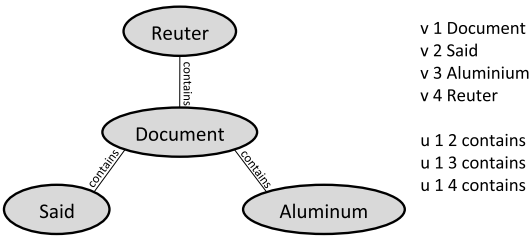

Figure 21. Sample Graph g2

```
Match Cost = 4.000000
Mapping (vertices of larger graph to smaller):
 1 -> 1
 2 -> 3
 3 -> 2
 4 -> 4
 5 -> deleted
```

Figure 22. Output for Transforming g1 to g2

Step 1 : Obtain the two representative substructures Rs1 and Rs2.

Step 2 : Find the largest substructure of the two and initialize it RS2 and the smaller one as RS1.

Step 3 : Compute the difference in the size between the two substructures by the formula

(v2 –v1) + (e2 – e1)

where

v2 and e2 are the number of vertices and edges of Rs2

v1 and e1 are the number of vertices and edges of Rs1.

Step 4 : Compute the match cost between Rs1 and Rs2 using Graph match module. Let M.C be the cost of transforming Rs2 to Rs1.

Step 5 : If ***(M.C <= ((v2 –v1) + (e2-e1)))***

Rs1 and Rs2 are ***similar*** substructures

else

Rs1 and Rs2 are ***not similar*** substructures

Figure 23. Algorithm to find if two representative substructures are similar

n is the total number of substructures in folder f_j

The numerator of this term denotes the frequency of representative substructure RS in all the folders in the training set except the one it was generated from. This is computed by calculating the frequency of RS across all the folders in the training set and then discounting its occurrence frequency in the folder it was generated from. Checking whether RS exists in all other folders is not straight forward because RS might be a part of the representative substructures in other folders or the same words constituting RS might not be in the same order in the representative substructures of other folders. Inspecting whether a representative substructure occurs, as a whole or part of another substructure, in other folders is done using the *graph match* module of Subdue. The graph match module takes two graphs as input and computes the cost of transforming the largest of the input graphs into the smaller graph. The cost is computed by summing up the number of operations to be done on the larger graph, such as adding or deleting a vertex label or an edge label, to map it to the larger graph. The output of the graph match module comprises of the vertex mapping of largest graph to the smaller graph along with the cost. For example, consider transforming graph g1 to graph g2 as shown in Figures 20 and 21.

As illustrated, graph g1 has got 5 vertices and correspondingly 5 edges while graph g2 has got 4 vertices and edges. Vertices of g1 (larger graph) are mapped to vertices of g2 along with the edges. The output of transforming g1 to g2 is shown in Figure 22. A cost of 2 is assigned to change the labels of vertices 2 and 3 of g1 (Rafael, Nadal) to vertices 2 and 3 of g2 (Said, Aluminum), cost of 1 to delete the extra vertex 5 (Roger) and a cost of 1 to delete the extra edge 'u 1 5 contains'. Therefore, the graph match module computes a cost of 4 for transforming g1 to g2.

The graph match module is used to examine whether representative substructure RS, that has been generated in folder f_{RS}, exists in other folders too. The representative substructures in other folders are considered to be similar to RS even if they contain RS as a subgraph in them. The algorithm shown in

Figure 23 has been developed to find if two representative substructures are similar.

The algorithm is initiated by finding the larger of the two subgraphs/substructures given as input. The largest subgraph is mapped to the smaller one as explained before (by renaming and deleting edges and nodes). The difference between the sizes of the two substructures are found by the formula $(v2 - v1) + (e2 - e1)$. The match cost of transforming one substructure to another is computed using the graph match module. The difference in size is compared with the match cost computed.

If the match cost is lesser or equal to the difference in size of the two substructure, then the two substructures are considered to be similar as both consist of the same vertex labels and edge labels. If the match cost is higher than the difference in the size of the two substructures, then the two substructures are not similar as labels of the vertices or edges are not the same. Using this algorithm, substructures similar to the one in question are generated and correspondingly, the frequency of the substructure is computed by adding the frequencies of all the similar substructures in other folders.

The denominator of the term $FRS(A, RS)$, as shown in equation 11, is the sum of the frequencies of all the representative substructures in all the folders of the training set. This term determines the common representative substructure RS occurs across all the folders. The importance of RS is inversely proportional to its commonality in occurrence across folders.

5.2.4 *Inverse Folder Frequency.* This term determines the number of folders in which representative substructure RS occurs. The intuition is that, if a representative substructure occurs in many document classes, then it is not a good discriminator and it should be ranked lesser than the ones which occur in fewer document classes. This measure is computed while calculating $FRS(A, RS)$ by maintaining an index of all the folders which contains representative substructures similar to RS. The IFF term provides a high value for common representative substructures and low value for unique representative substructures. So when the inverse of IFF(RS) is considered, it provides a high value for rarely occurring substructure and low value for a representative substructure that exists across many folders. For example, if RS exists in 3 folders (including the folder it was generated from), then the value for $\frac{1}{IFF(RS)^2}$ would be $0.1111(1/9)$. Whereas, a RS that exists only in the folder it was generated from would have a value of 1.

5.2.5 *Size of the Representative Substructure.* The final term in the global rank formula in Equation 9 is the *size of the representative substructure*. Relatively large sized frequent substructures signify greater similarity among the documents in a class. The size of the representative substructure is compared with the largest substructure in the global list. Based on its relative size, a weight is computed for the representative substructure. Therefore, a representative substructure that compares well with the size of the largest substructure, is assigned a higher weight when compared to smaller substructures. A representative substructure gets a higher rank due to:

(1) It occurs frequently across the same folder it was generated from
(2) It occurs less frequently across all the folders in the training set except the one it was generated from
(3) The size of the representative substructure compares well to the largest substructure in the global list

Once all the representative substructure are ranked, they are ordered based on their ranks. The list of ordered substructures are then used during classification. The classification process is explained in the following section.

5.3 Classification

The ranked substructures are used for classifying the incoming unknown documents. In order to assign an appropriate label to this document, it is compared with the ranked substructures. As with the generation of representative substructures, inexact graph match is used for comparing the unknown document with predefined representative substructures. Each ranked substructure is embedded into the test document to create a forest of two graphs: 1) the test document represented as a graph, and 2) the graph of the representative substructure.

The classifier is used to generate representative substructures from the forest of graphs with a minimum size set to the size of the representative substructure embedded into the test document. The list of substructures generated by the classifier are checked for its occurrence frequency. If the occurrence of the

substructure (which is the embedded representative substructure) is greater than 1, then the test document has got an instance of the representative substructure in it. It denotes that the test document contains the words that make up the representative substructure, along with the same relationship. It also means that the test document comprises of the frequent words that represent the document class in the form of the representative substructure. Hence, the test document is filed to the same class with the highest ranked substructure match signifying higher correlation with the class contents.

6. EXPERIMENTAL EVALUATION

This section presents the experimental analysis and results to validate our premise that words in document classes exhibit relationships and these patterns can be beneficially used to learn and aid in classification of unknown documents. The applicability of this approach across *single* as well as *multiple*folders for diverse documents namely text, emails, and web pages have been considered. The accuracy of the classifier on these domains is very good and is consistent. The experimental setup and a brief description of the data set used is also provided. A brief overview of the system implementation and the details of the configuration parameters is presented below.

6.1 Implementation Details

The *InfoSift* framework has been designed in Perl. It must be noted that as the Subdue discovery algorithm is implemented in C, the choice of using an interpreted language for developing the various modules of the document classification system does not hurt the overall performance. The prototype system is an amalgamation of separate yet inter-related set of modules namely, *document pre-processing*, *graph generation*, *substructure extraction*, *substructure pruning*, *representative substructure ranking* and *classification*.

The input to the training module is a set of one or more document classes, along with various parameters for graph generation and pattern discovery. The parameters are provided in a configuration file comprising of options for – split for cross validation, choice of graph representation, beam value, etc. The system pre-processes the classes in the training set, generates graphs, computes the various class characteristics and invokes the substructure discovery algorithm. The output generated is pruned and, in the case of *multi-folder* classification ranked across all the folders in the training set to produce the *global rank list*. This list is then used during classification of the test document. The outcome of the classification along with the output of each module are logged for analysis. In the discussion that ensues, we will briefly describe some of the implementation aspects of important modules and details about the different configuration parameters.

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) **Number of Document Classes:** The total number of document classes fed to the training model.

(2) **Name of Document Classes:** The names of the document classes or folders that contain the documents to train the classifier.

(3) **Graph Representation:** Various graph representations such as star and tree have been proposed for different domains. Each graph representation has been assigned an unique number id.

(4) **Training/Test Set Split:** The document classes containing different documents provide information for training the classifier. The percentage of the class sample to be used for training can be specified as a ratio using this parameter in the configuration file. A training/test split of 80:20 and 60:40 has been used i.e., 80% or 60% of the documents in the class are used to train the documents in order to

generate representative substructures that represent the documents and the remaining 20% or 40% of the documents are used for classification.

(5) **Feature Subset Selection:** The top *f*% of the features representing the document class to be selected during the pre-processing of the folder content (f values of 70, 80, and 90 have been evaluated).

(6) **Random/Sequential Generation:** The option of choosing the *first* n% of the documents in the class or *randomly* choosing n% documents to act as the training set can be specified using this parameter.

(7) **Seed:** In case of random selection of documents for the training set and test set for classification, a seed value can be provided for the randomized generation. If left unspecified, the system supplies a default value for generation process.

(8) **Log File:** The file name to log the results of the outputs of each modules during processing and classification. The logged information also contains values of the parameters that were specified in the configuration file. Additionally, information regarding the substructure generation and the representative substructure that matched with the test document are logged for further analysis. In case a log file is not specified, a default name derived from the class names and other attributes is used.

(9) **Graph File:** The documents in the training set are represented as a forest of graphs for the substructure discovery process. The file name to store the forest of graphs is specified using this parameter. If a file name is not specified, a default value is assigned to the file based on the class names and its attributes.

(10) **Substructure Output File:** The file name to store the output of the substructure generation process. It contains the values of the input parameters to the substructure discovery algorithm in the Subdue system along with the top best substructures that were extracted from the input training data set. This file is used during the pruning phase in order to filter out the repetitive substructures that were extracted. A default filename is taken if no name is specified.

(11) **Substructure Discovery Threshold:** The amount of inexactness that is permissible during substructure discovery process is specified using this parameter. This value can be specified by the user else it is calculated as explained in Equation 7. A value of 0.1 has been used for our experiments to compare the effect of inexact graph match on classification with exact graph match.

(12) **Classification Threshold:** This value represents the threshold during the classification of the test document. As the classifier searches for the occurrences of the representative substructure inside the graph representation of the test document, an amount of inexactness is also allowed to group similar instances of patterns just as in substructure discovery. This value can be specified by the user else the same value as substructure discovery threshold is used as default. A value of 0.05 have been used for our experiments.

(13) **Minsize:** The minimum size of the substructures generated by the substructure discovery process can be constrained above a certain value by this parameter. If not given, it is computed based on the graph type used.

(14) **Beam:** The value of the *beam* for the substructure discovery algorithm. Values of the 2, 4, 8 and 12 have been used for the experiments. If the values are unspecified, then a value of 4 is used for small classes and a value of 8 has been used for medium and large classes by default.

(15) **Prune:** This parameter can be turned on or off depending upon whether the output of the discovery process containing the list of best substructures needs to be pruned or used as is for classification. The default is to prune the substructures.

6.1.2 *Graph Representation and Generation.* The documents that are collected as training set from the document class are used to derive substructures to represent the respective class. For this reason, the documents are converted into graphs that act as inputs to the substructure discovery process. The graph generator developed is capable of generating graphs for various domains such as text, email, and web pages. For processing emails, the Perl packages *Mail::Internet* and *Mail::Address* are used to extract the header and body information respectively. *HTML::TokenParser* package is used for processing web pages and deriving the necessary information from the HTML tags in the pages.

Associative arrays in Perl have been used to store the term-frequency pairs of the features in the training set. The documents that form the training set are used to construct a *global hash* of term occurrences across all the documents in the class. This set of terms are pruned based on the feature subset selection percentage that is to be retained. During the construction of the graphs for sample documents, only those terms that occur in the global hash after pruning are considered. Class statistics such as document class size and average document size in the class are also computed and logged for substructure discovery during graph generation.

6.1.3 *Substructure Discovery.* The pattern discovery pattern is handled by the Subdue substructure discovery algorithm. The input to this system such as *threshold*, *minsize*,*graph input file*, *output file name*, etc. are specified in the configuration file. The output of the substructure generation process is written to the file which is processed to prune substructures and generate the representatives of the class under construction.

6.2 Representative Substructure Pruning and Ranking

In the case of *multi*-folder classification, the representative substructures are compared to eliminate those that are similar in terms of substructure size, MDL and frequency and differ only in their description of an edge or vertex label. The list of best substructures from the substructure generation output file is analyzed to discount the similar substructures as per our definition of similarity explained in Section 5.1. In addition, certain large-sized classes substructures that are highly infrequent are pruned as well.

The substructures (after pruning) from each folder in the training set are merged together into a single list and ranked against each other to position them in an ordinal scale based on their representativeness. Associative arrays are used to store the representative substructures. The *Data::Dumper* module is used to save the information in the form of hash data structure. The ranked substructures are sorted using the sort function provided by Perl. For each unique rank (key of the hash), information about the corresponding substructure such as substructure name, folder it belongs to, rank value, size, $FRS(f, RS)$, $FRS(A, RS)$, $IFF(RS)$ are stored. During classification, the classifier tries to match the test document with the substructures in the sorted order. Once a match is found, the classifier stops further comparison with representative substructures and assigns the same label as that of the representative substructure that it matched.

6.3 Experimental Results

The results of classification experiments on different domains such as text, emails, and web page repositories are discussed here. The experiments have been carried out on Intel Xeon CPU 2.80 Ghz dual processor machines with 2GB memory. Exhaustive experiments on a large number of classes with diverse characteristics (different document class size, dense, sparse classes, etc.) have been carried out to study the effect of parameters on classification of unknown test documents in a single as well as multiple folder environment.

6.3.1 *Classification for Text Repositories.* We have used the Reuters-21578 corpus[4], 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

[4]http://www.daviddlewis.com/resources/testcollections/reuters21578/

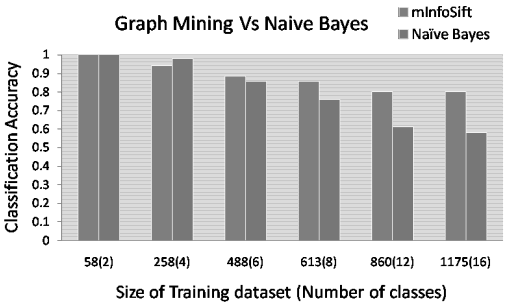

Figure 24. Multi-Folder Classification: *InfoSift* vs. Naive Bayes

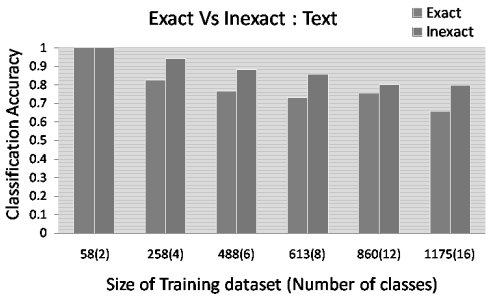

Figure 25. Multi-Folder Classification: Inexact vs. Exact Match

naive Bayesian classifier, implemented in the *Bow* library developed by Andrew McCallum[5]. Each of the experimental results are now considered in detail.

6.3.2 *Graph Mining vs. Naive Bayes.* Figure 24 shows the comparison between the performance of our approach with the Naive Bayesian one for *multi*-folder classification. Both the approaches have been tested on different training set size from multiple folders (2 to 16 folders) containing small, medium and large classes. The classification of the Graph Mining approach is consistently better than the Bayesian approach. Both the classifiers perform well with small number of folders such as 2, 4 and 6. With the increase in the number of folders in the training set, classification accuracy of both the approaches decreases due to the increasing number of misclassified test documents. It is clear that as the number of classes increased beyond 6, the difference between naive Bayes and *InfoSift* increases significantly (as much as 50% for 16 classes and containing 1175 documents).

Some of the reasons as to why a test document might get wrongly classified is listed below.

Lack of adequate data in test document. Some of the test documents from the class did not contain enough information for it to be classified to any specific class. Test documents with minimum information tend to match with substructures of minimum size, which occur in most of the documents in various classes and hence gets a low rank. This leads to the test document being classified to the wrong folder.

Folders with lot of heterogeneous documents. The input data sets used for experimental analysis contain classes with documents already labeled and classified. Sometimes the classes contains documents dealing with diversified information, i.e., the documents under the same class are very heterogeneous. This leads to large substructures with very low frequency returned as the best substructures by the discovery algorithm. These substructures are ranked lower than the substructures of smaller size with high frequency from other folders. Therefore, small substructures with very high frequency in folders tend to get a higher rank than the large substructures with very low frequency of the heterogeneous folder. So when the test document, which actually belongs to this heterogeneous folder, is tried to classified, it is likely to get classified to the wrong folder due to presence of highly ranked small substructures of other folders.

6.3.3 *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 in Figure 25 for *multi*-folder classification.

As evident from result, inexact graph consistently performs better than exact graph match. The performance of exact and inexact graph match is similar for smaller number of folders but with the increase in the number of folders, the difference in performance becomes more clearer. The training set of classes

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

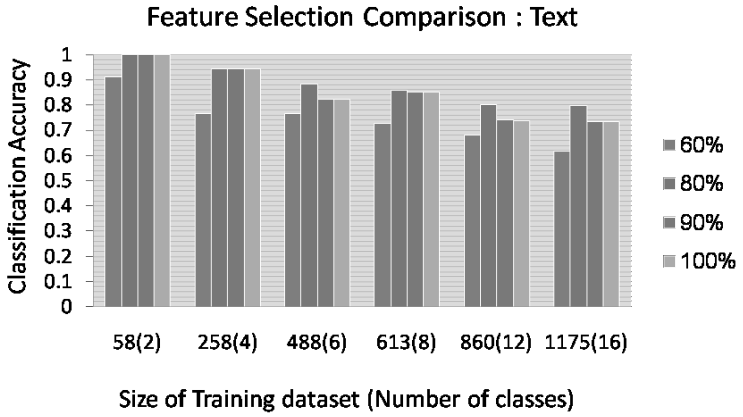

Figure 26. Multi-Folder Classification: Effect of Feature Set Size

comprises of both small classes and large classes. A reason as to the better performance of inexact graph match is because it is able to group similar instances that vary slightly even in the absence of large training data set in the case of small folders.

6.3.4 *Feature Subset Selection.* 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 Section 4.2.3. We have used three different sizes of the feature set by extracting the top 100%, 90%, 80%, and 60% 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 Figure 26 for *multi*-folder classification.

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 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. We have used 80% for most of the experiments as that value tends to provide as good results as 90% and 100% and even better sometimes.

6.3.5 *Comparison between Tree and Star Representation.* Toward *multi*-folder classification, we have proposed two graph representations – *Star* and *Tree*, that have been used to represent the training and test documents to the graph mining system, Subdue. Figure 27 show the results of the comparison of the two graph representations on the classification accuracy. The tree representation shows significantly better accuracy over the star representation. The tree representation exhibit better structural relationships between the words than the star representation. The better performance of the tree structure is attributed to the presence of extra layer in its structure (the first layer contains only the root node followed by another layer containing vertices of different components that make up the document like title, body, etc. and a third layer of vertices with all the features attached to the corresponding vertices in the second layer). The star has got only two layers: the root forms the first and the rest of the vertices form the second. Note that bcc and attachments have not been used in the current framework. They may provide additional discrimination which is likely to improve the accuracy further.

6.3.6 *Prune vs. No Pruning.* The substructure discovery process generates large number of sub-structures from the training data set. But all of them are not likely to contribute towards classification.

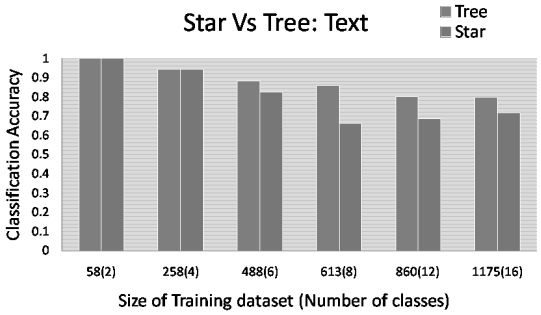

Figure 27. Multi-folder Classification: Star vs. Tree representation

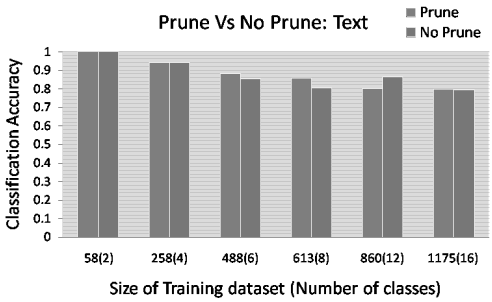

Figure 28. Multi-folder Classification: Pruning vs. Non Pruning comparison

Furthermore, the cost of processing all of the substructures only increases. Due to these reasons, a pruning option was developed for *multi*-folder classification where instances of slightly varying substructures were analyzed and pruned except for a single instance. Figure 28 illustrates the results on classification accuracy with and without pruning. The pruning approach turns out to be better since lesser number of substructures were retained that still exhibited their corresponding class characteristics. The more the number of substructures retained, the more the chance for a test document to get misclassified.

With these experiments on the text corpus, we can claim that *multi*-folder classification outperforms a conventional text classifier. 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 expected and suggested earlier, inexact graph match yields better classification results when compared to exact graph match. A tree representation with its superior structure showed better classification ability than the star representation consistently. Pruning of the substructures aided towards a better classification result. With the knowledge of these results, we will describe our results for the domain of email classification.

6.4 Classification of Email Corpora

Initially, we 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.

Additionally, to show that our *multi-folder* classification approach is specifically consistent across diverse data sets, experiments were performed using Enron Email Data set (http://www.cs.cmu.edu/ enron/). The Enron email data set has been collected and prepared by the CALO [6] project. It comprises of data in the form of emails from about 150 users organized into folders. The email folders in this data set have been cleaned and organized before it can be used for training the classifier. Some pre-processing steps include removing the non-topical folders (folders containing email messages regardless of their contents such as Inbox, Sent, Trash, Drafts, etc.), removing folders that are too small (does not contain any messages) or too large (contain more than 600 messages), etc. Experiments were conducted on both these data sets in order to show that our approach is compatible to different types of messages from the same domain. The results of our experiments are presented in the following sections.

6.4.1 *Graph Mining 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 heterogeneous content as well as on sparse folders. The Bayesian classifier compares poorly and produces many false positives. In cases where naive Bayes performs better, it is only very marginally better and always for small number

[6] More information at http://www.ai.sri.com/project/CALO

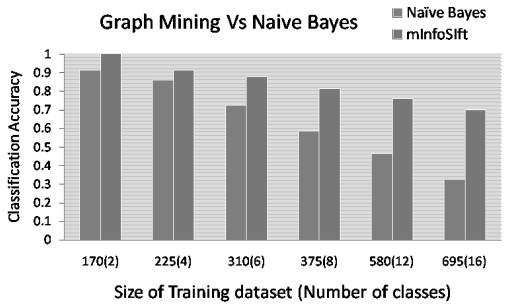

Figure 29. Multi-Folder Classification: *InfoSift* vs. Naive Bayes (Listserv data set)

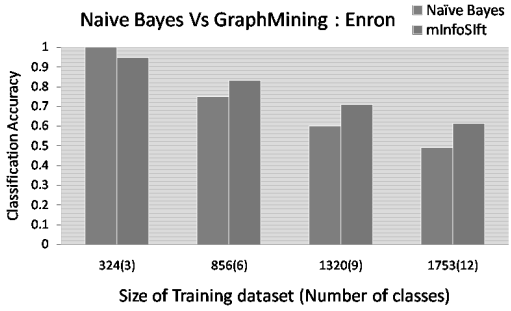

Figure 30. Multi-Folder Classification: *InfoSift* vs. Naive Bayes (Enron data set)

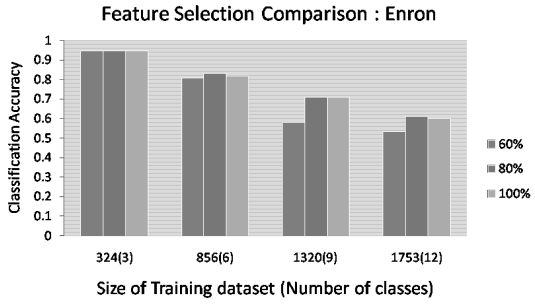

Figure 31. Multi-Folder Classification: Effect of Feature Set Size (Listserv data set)

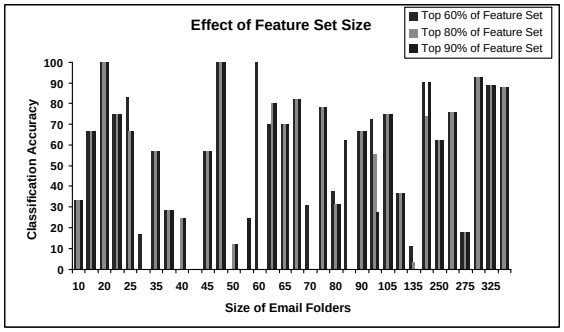

Figure 32. Multi-Folder Classification: Effect of Feature Set Size (Enron data set)

of classes. One of the remarkable features of *InfoSift* is the presence of a very low rate of false positives. As is evident from the Figure 29 for *multi*-folder classification, *InfoSift* performs much better than naive Bayes. *InfoSift* was consistent in classifying incoming emails, but naive Bayes was unable to classify with good accuracy for many input folders.

The performance of *InfoSift* for *multi*-folder classification is consistent across the Enron data set as well. Though the difference in performance of both the classifiers for the Enron data set is not as distinguishable when compared to that over Listserv data set, our approach out performs Naive Bayes approach consistently. The results for Enron data set is presented in Figure 30.

As alluded to previously, 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 *patterns* of word occurrences and avoids this problem. As observed in the case of text classification, naive Bayes performs slightly better than *InfoSift* when the number of classes is very small. We believe that the use of a higher beam value will improve classification even for small number of classes.

6.4.2 *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 Figures 31 (for *multi*-folder classification on Listserv) and 32 (for *multi*-folder classification on Enron). Similar to 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,

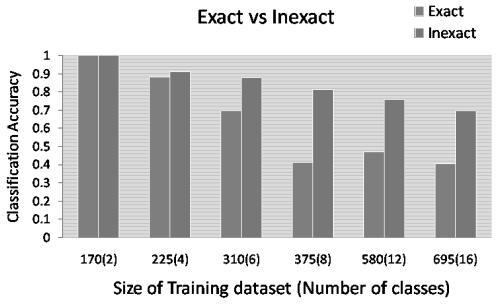

Figure 33. Multi-Folder Classification: Exact vs. Inexact Substructure Discovery (Listserv data set)

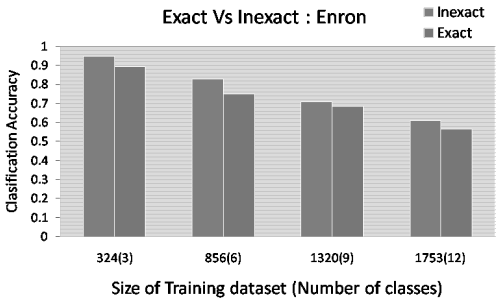

Figure 34. Multi-Folder Classification: Exact vs. Inexact Substructure Discovery (Enron data set)

as in the case of text classification, the variation was observed more in the case of small and medium-sized folders. The increase in classification accuracy can be attributed to the removal of noise in the form of irrelevant features. Feature sets comprising the top 70% and 90% performed similarly. Based on this, 80% has been used in all comparison experiments.

6.4.3 *Exact 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 Figures 33 and 34 for *multi*-folder classification over Listserv and Enron data sets, respectively.

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 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.

6.4.4 *Tree vs. Star Representations.* Experiments were conducted to study the effect of different graph representations on *multi*-folder email classification. The tree representation showed significantly better performance in classification as shown in Figure 35. This figure corresponds to the Listserv data set collected for the *InfoSift* framework.

The difference in performance between star and tree representation is greater in email domain rather than in text classification. This is attributed to the enhanced structural information exhibited by email messages when compared to text documents. The tree representation represents emails, due to their structural information, in a better way than the star representation. This is reinforced with the results of the experiments conducted on Enron data set. Figure 36 illustrates the results of Tree and Star representations on Enron email data set. Evidently, tree representation has a better performance in classification than the star representation.

With the above experimental results in hand, we can draw conclusions on the approach of graph mining (used by the *InfoSift* system) for the email domain. The accuracy 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

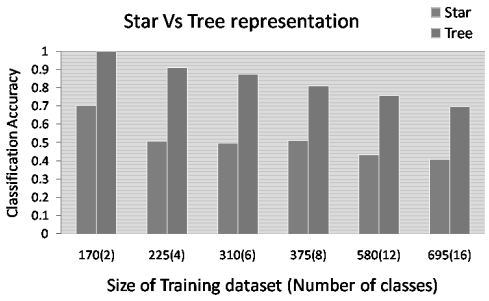

Figure 35. Multi-Folder Classification: Star vs. Tree representation (Listserv data set)

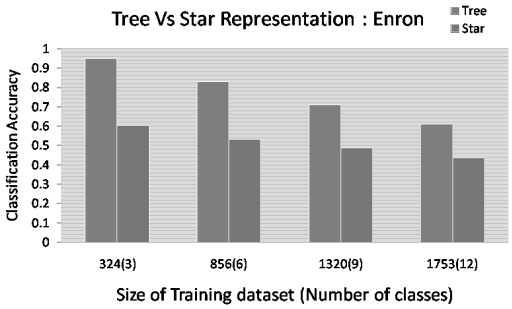

Figure 36. Multi-Folder Classification: Star vs. Tree representation (Enron data set)

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 accuracy 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.

6.5 Web Page Classification

For evaluation of web pages, we have conducted experiments on web collections called the *K-Series* [7]. The K-series consists of around 2,300 documents that belong to 20 different categories such as Art, Entertainment, Music, etc. A random selection of 890 documents have been used for experimental evaluation, and similar to the text classification methodology, Accuracy and Error rate are considered as performance metric.

The results of the experiment using the K-Corpus are presented below. Extensive experiments were conducted to determine the performance of the classifier. Figure 37 shows *multi*-folder classification of the K-Corpus. It can be observed that *InfoSift* provided consistent classification accuracy that is equal to or much better than naive Bayes across most categories.

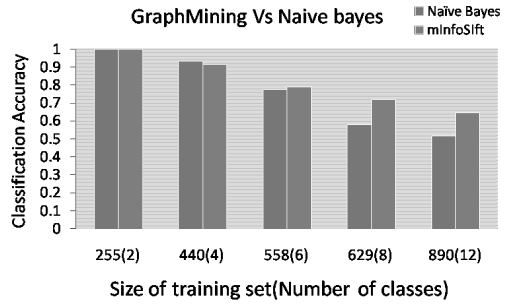

Figure 37. Multi-Folder Classification: Performance of *InfoSift* for the K-Corpus

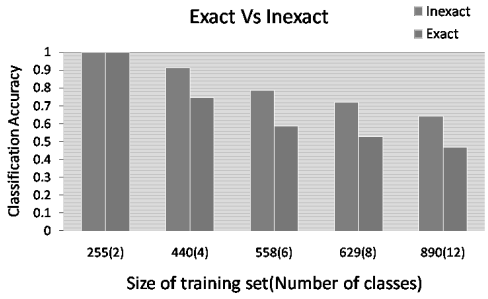

Figure 38. Multi-Folder Classification: Inexact vs. Exact Graph Match for the K-Corpus

[7] publicly available at ftp://cs.umn.edu/users/boley/PDDPdata/

For *multi*-folder classification, the strength of our approach lies in ranking the substructures globally across all the document classes in the training set which in turn produces lower number of false positives when compared to Naive Bayes. With the increase in the number of distinct folders in the training set, the error rate also increases for both the approaches but when compared with each other, our approach exhibits a superior performance.

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 Figure 38 for *multi*-folder classification. The performance due to inexact graph match is significantly better justifying its use both while learning patterns and during classification.

6.6 Desiderata

We proposed a novel way of classification of different types of documents (text, email, and web pages) by exploiting their underlying structure as well as content. Subdue was chosen as it matched our requirements of inexact match which is not supported in other graph mining approaches/systems. Use of representative substructures (instead of bag of words) seems to characterize documents better. Deriving all the parameters from the data set was important as it has the ability to adapt to data set characteristics. To validate our intuition and premise, we have carried out exhaustive experiments across three different types of documents and presented the results. The consistent performance of the *InfoSift* classifier both for single- and multi-folder classification has validated our hypothesis about the feasibility and accuracy of the proposed novel approach for various types of documents such as text, email and web pages.

7. CONCLUSIONS AND FUTURE WORK

This paper proposed an innovative approach for document classification that leverages structural properties of a document. Our approach demonstrated how graph mining techniques, developed for a different purpose, can be effectively adapted for classification. In fact, we have proposed a framework and its components along with techniques for fine-tuning the parameters of this framework for the classification of various types of documents (text, email, and web pages). Our premise that patterns in a class of documents or within emails of a folder can be extracted and used for classification is substantiated by the results shown in this paper. Our approach overcomes the limitations of conventional techniques that cannot take into account the structural properties and their relationships on the components of the documents (terms in our case).

We have developed a prototype that automates the task of classification using our proposed approach. We have experimented with three different types of documents (email, text, and web pages) to validate the effectiveness as well as the generality 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. With the work presented in the paper, we have developed the underpinnings of the usage of graph mining techniques for the task of classification. The feasibility of the novel graph mining-based approach for classification has been established by the consistent acceptable performance of the classifier over different text-based domains. Various parameters that affect classification have been identified and analyzed in detail.

Although the performance of the classifier is acceptable (as most of the time-consuming work is done as pre-processing), more work needs to be carried out to make it more efficient. Some of the enhancements that can be carried out are outlined below.

7.1 Future Work

In this work, we have used Subdue as a black box without attempting to change the internals for improving performance. We believe that its internals can be modified to improve performance significantly. Subdue and other mining approaches do no allow weights to be attached to edges or vertices. For example, 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. This approach can also be used beneficially for emails. At present there is no provision to associate different weights to different parts of a graph to indicate a higher significance based

on weights. Also, Subdue provides a '$setcover$' evaluation metric for determining repetitive substructures in the input graph which may be used to improve performance.

Our feature selection does take into account word occurrences in different parts of the document. This may be significant as word occurrences in a header/heading may be more important than others. Also, we have not explored the combination of usage of only words for some components (cc, bcc etc.) and structures for others. It may be possible to combine the best of both for improving accuracy and also improving efficiency.

For some applications (e.g., email classification), the adaptation of the classifier to changes in email folder characteristics is critical for achieving continued good classification accuracy. In the current implementation, incremental learning is not performed. This is likely to be critical for the email domain, as new emails are added to a folder resulting in changes to folder characteristics. Experiments presented in this paper 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 several options for incremental learning:

Batch Learning. The representative substructures of a folder are recomputed after every 'n' deletions or additions to the folder.

Selective/incremental Learning. Rather than *recomputing* the representative substructures, only those that correspond to the new emails are computed and the set of current representative substructures are updated. This may involve updating the ranks of certain substructures that are already part of the set.

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 substructures 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 need to be analyzed to improve the current scheme or consider the use of other techniques for feature extraction. Also, a comparison of the system with classifiers besides naive Bayes will reveal useful insights into enhancements required for better classification. The future work also includes the development of a graphical user interface for the email classifier to be coupled or used as a plug-in with an email agent.

Performance of classifiers is also an important factor. This paper mainly 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 practical for real-world applications. A redeeming aspect is that the generation of representative substructures can be pre-computed and only the classification of incoming emails need to be performed in real time.

While this work establishes a framework for classification using graph mining for text classification, other application domains remain to be researched. For example, proposed techniques can be applied to a 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 the request for a new patent can be matched against previous patents to determine if anything similar is already present in the patent database. Our approach can also be used for detecting plagiarism in large text documents.

In summary, we believe the adaptation of graph mining approach for classification is useful, effective, and opens up new possibilities and research directions that is novel and different from contemporary classification approaches.

REFERENCES

AERY, M. 2004. Infosift: Adapting Graph Mining Techniques for Document Classification. M.S. thesis, The University of Texas at Arlington.

AERY, M. AND CHAKRAVARTHY, S. 2005a. eMailSift: Email Classification Based on Structure and Content. In *ICDM*. 18–25.

AERY, M. AND CHAKRAVARTHY, S. 2005b. InfoSift: Adapting Graph Mining Techniques for Text Classification. In *FLAIRS Conference*. 277–282.

AGARWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. *Proceedings of 1993 ACM SIGMOD Conference*, 207–216.

APTE, C., DAMERAU, F., AND WEISS, S. M. 1998. Text mining with decision trees and decision rules. *Conference on Automated Learning and Discovery*.

APTE, C., F.DAMERAU, AND WEISS, S. 1994. Towards language independent automated learning of text categorization models. *In the Proceedings of the 17th Annual ACM/SIGIR conference*, 23–30.

ATTARDI, G., GULLI, A., AND SEBASTIANI, F. 1999. Automatic we page categorization by link and context analysis. *In Chris Hutchison and Gaetano Lanzarone (eds.), Proc. of THAI'99*, 105–119.

BAKER, L. D. AND MCCALLUM, A. K. 1998. 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)*, 96–103.

BOONE, G. 1998. Concept features in re:agent, an intelligent email agent. *Proc. Agents*, 141–148.

BRUTLAG, J. D. AND MEEK, C. 2000. Challenges of the email domain for text classification. *Proceedings of ICML*, 103–110.

BUNKE, H. AND ALLERMAN, G. 1983. Inexact graph match for structural pattern recognition. *Pattern Recognition Letters*, 245–253.

BUNKE, H. AND SHEARER, K. 2001. A graph distance metric based on maximal common subgraph. *Pattern Recognition Letters*, 753–758.

CATLETT, J. 1991. Megainduction: A test flight. *Proc. International Conference on Machine Learning*, 596–599.

CHAKRAVARTHY, S., VENKATACHALAM, A., AND TELANG, A. 2010. A graph-based approach for multi-folder email classification. In *ICDM*. 78–87.

CLARK, P. AND NIBLETT, T. 1989. The cn2 induction algorithm. *Machine Learning*, 261–283.

COHEN, W. W. 1995. Text categorization and relational learning. *In the Twelfth International Conference on Machine Learning (ICML'95)*, 124–132.

COHEN, W. W. 1996. Learning rules that classify e-mail. *Proceedings of AAAI-1996 Spring Symposium on Machine Learning in Information Access*, 124–143.

COHEN, W. W. AND SINGER, Y. 1995. A simple, fast, and effective rule learner. *In Proceedings of the National Conference on Artificial Intelligence*, 335–342.

COHEN, W. W. AND SINGER, Y. 1996. 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*, 307–315.

COOK, D. J. AND HOLDER, L. B. 1994. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research 1*, 231–255.

COOK, D. J. AND HOLDER, L. B. 2000. Graph based data mining. *IEEE Intelligent Systems 15(2)*, 32–41.

CRAWFORD, E., KAY, J., AND MCCREATH, E. 2001. Automatic induction of rules for e-mail classification. *Proceedings of the Sixth Australasian Document Computing Symposium, Coffs Harbour, Australia*.

FRAWLEY, W., PIATETSKY-SHAPIRO, G., AND MATHEUS, C. 1992. Knowledge discovery in databases: An overview. *AI Magazine*, 213–228.

GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research 3*, 1157–1182.

HEFLMAN, J. AND ISBELL, C. 1995. Ishmail: Immediate identification of important information, AT&T labs.

JOACHIMS, T. 1998. Text categorization with support vector machines: Learning with many relevant features. *ECML*, 137–142.

KETKAR, N. S., HOLDER, L. B., AND COOK, D. J. 2005. Subdue: compression-based frequent pattern discovery in graph data. In *OSDM '05: Proceedings of the 1st international workshop on open source data mining*. ACM Press, New York, NY, USA, 71–76.

KIRITCHENKO, S., MATWIN, S., AND ABU-HAKIMA, S. 2004. Email classification with temporal features. In *Intelligent Information Systems*. 523–533.

KOLLER, D. AND SAHAMI, M. 1997. Heirarchically classifying text using very few words. *In the 14th International Conference on Machine Learning (ICML'97)*, 170–178.

KURAMOCHI, M. AND KARYPIS, G. 2001. Frequent subgraph discovery. *IEEE International Conference on Data Mining*, 313–320.

LAM, W. AND HO, C. 1998. 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)*, 81–89.

MASAND, B., LINOFF, G., AND WALTZ, D. 1992. Classifying news stories using memory based reasoning. *In the 15th Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'92)*, 59–64.

MCCALLUM, A. K. AND NIGAM, K. 1992. 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)*, 59–64.

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

MOULINIER, I., RASKINIS, G., AND GANASCIA, J. 1996. Text categorization: a symbolic approach. *In the Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, 87–99.

NG, H. T., GOH, W. B., AND LOW, K. L. 1997. 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)*, 67–73.

PAYNE, T. AND EDWARDS, P. 1997. Interface agents that learn: An investigation of learning issues in a mail agent interface. *Applied Artificial Intelligence*, 1–32.

RENNIE, J. D. M. 2000. ifile:an application of machine learning to e-mail filtering. *Proceedings KDD Workshop on Text Mining,Boston, MA*, 92–100.

RISSANEN, J. 1989. Stochastic complexity in statistical enquiry. *World Publishing Company*.

SAHAMI, M., HECKERMAN, D., AND HOROVITZ, E. 1998. A bayesian approach to filtering junk e-mail. *AAAI-98 Workshop on Learning for Text Categorization*, 98–105.

SALTON, G. 1991. Developments in automatic text retrieval. *Science 253*, 947–980.

SCHENKER, A., LAST, M., BUNKE, H., AND KANDEL, A. 2003. Classification of web documents using a graph model. *Seventh International Conference on Document Analysis and Recognition*, 240–244.

SEGAL, R. B. AND KEPHART, J. O. 2000. Swiftfile: An intelligent assistant for organizing e-mail. *Proceedings of AAAI 2000 Spring Symposium on Adaptive User Interfaces*, 107–112.

TZERAS, K. AND HARTMAN, S. 1993. Automatic indexing based on bayesian inference networks. *In the 16th Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'93)*, 22–34.

VAPNIK, V. N. 1982. *Estimation of Dependences based on Emperical Data [In Russian]*. Nauka, Moscow, 1979. (English Translation Springer Verlag, New York, 1982).

VAPNIK, V. N. 1998. *Statistical Learning Theory*. John Wiley & Sons.

VENKATACHALAM, A. 2007. m-InfoSift: A Graph based Approach for Multiclass Document Classification. M.S. thesis, The University of Texas at Arlington.

WEINER, E., PEDERSON, J. O., AND WEIGEND, A. S. 1995. A neural network approach to topic spotting. *In the Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95)*, 317–332.

WHITTAKER, S. AND SIDNER, C. 1996. Email overload: exploring personal information management of email. *Conference Proceedings on Human Factors in Computing Systems*, 276–283.

YANG, Y. 1994. 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)*, 13–22.

YANG, Y. AND CHUTE, C. G. 1994. An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems (TOIS) 12(3)*, 252–277.

YANG, Y. AND LIU, X. 1999. A re-examination of text categorization methods. *ACM SIGIR*, 42–49.

Sharma Chakravarthy is Professor of Computer Science and Engineering Department at The University of Texas at Arlington (UTA). He is the founder of the Information Technology laboratory (IT Lab) at UTA. His research has spanned semantic and multiple query optimization, complex event processing, social network analysis, and web databases. He is the co-author of the book: Stream Data Processing: A Quality of Service Perspective (2009). His current research includes social network analysis, information integration, web databases, recommendation systems, integration of stream and complex event processing, and knowledge discovery. He is an ACM Distinguished Scientist and a Senior member of IEEE. He has published over 170 papers in refereed international journals and conference proceedings.

Aravind Venkatachalam works as Senior Member of Technical Staff at Salesforce.com in San Francisco, CA. Prior to joining Salesforce, he has held various engineering positions at Hi5.com and Ask.com. He completed his M.S. from University of Texas at Arlington and B.Tech from University of Madras in India. His current work and interests are Big Data, building highly scalable web systems and analytics.

Aditya is a researcher in the Information Management Group at IBM Research India since 2011. Prior to joining IBM, he finished his PhD from University of Texas at Arlington. He received his Masters from State University of New York at Buffalo. His current research interests include Spatio-Temporal Data Analytics, Information Management, and Analytics for Retail Banking.