2004

Feature Partitioning for the Co-Traning Setting

Vineet Chaoji

FEATURE PARTITIONING FOR THE CO-TRAINING SETTING

BY

VINEET CHAOJI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

ROCHESTER INSTITUTE OF TECHNOLOGY

July 2004

Feature Partitioning for the Co-training Setting

Thesis report submitted in partial fulfillment of the requirements of the degree Master of Science in Computer Science

July 2004

Vineet Shashikant Chaoji

Advisor : Dr. Ankur Teredesai

Reader : Dr. Roger Gaborski

Observer : Dr. Khalid Al-Kofahi

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: FEATURE PARTITIONING FOR THE CO-TRAINING SETTING

Name of author: VINEET CHAOJI
Degree: MASTERS OF SCIENCE
Program: COMPUTER SCIENCE
College: GOLISANO COLLEGE OF COMPUTING & INFORMATION SCIENCES

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, VINEET CHAOJI, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: _____ Date: 11/29/2004

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, _____, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: _____ Date: _____

Abstract

Supervised learning algorithms rely on availability of labeled data. Labeled data is either scarce or involves substantial human effort in the labeling process. These two factors, along with the abundance of unlabeled data, have spurred research initiatives that exploit unlabeled data to boost supervised learning. This genre of learning algorithms that utilize unlabeled data alongside a small set of labeled data are known as semi-supervised learning algorithms.

Data characteristics, such as the presence of a generative model, provide the foundation for applying these learning algorithms. Co-training is one such algorithm that leverages existence of two redundant "views" for a data instance. Based on these two views, the co-training algorithm trains two classifiers using the labeled data. The small set of labeled data results in a pair of weak classifiers. With the help of the unlabeled data the two classifiers alternately boost each other to achieve a high-accuracy classifier.

The conditions imposed by the co-training algorithm regarding the data characteristics restrict its application to data that possesses a natural split of the feature set. In this thesis we study the co-training setting and propose to overcome the above mentioned constraint by "manufacturing" feature splits. We pose and investigate the following questions:

1. Can a feature split be constructed for a dataset such that the co-training algorithm can be applied to it?

2. If a feature split can be engineered, would splitting the features into more than two partitions give a better classifier? In essence, does moving from co-training (2 classifiers) to k-training (k-classifiers) help?

3. Is there an optimal number of "views" for a dataset such that k-training leads to an optimal classifier?

The task of obtaining feature splits is approached by modeling the problem as a graph partitioning problem. Experiments are conducted on a breadth of text datasets. Results of k-training using constructed feature sets are compared with that of the expectation-maximization algorithm, which has been successful in a semi-supervised setting.

Acknowledgments

I would like to thank my committee members for giving me a rich Masters thesis experience. Without their efforts this thesis would not be as it stands today.

I would like to thank my advisor, Dr. Ankur Teredesai, for the constant encouragement and the positive attitude he infused during this thesis. He always had new ideas and suggestions regarding the applications of our work to various areas. As a graduate student at RIT, the Laboratory for Applied Computing (LAC) provided the ideal setting for undertaking research. I am thankful to Dr. Gaborski for nurturing this environment at LAC. This wonderful environment at LAC motivated us to put in the extra hours. I am very thankful to Dr. Khalid Al-Kofahi for taking the time from his busy schedule to be a part of my committee. His feedback was invaluable.

I would also like to thank Catalin Tomai for his interest in my thesis work and for all the meetings I had with him. The discussions with him have greatly contributed to my thesis. Finally, I would like to thank each and every member of the LAC for being a part of my thesis, by providing suggestions, comments and encouragement or by just being there to listen to me.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

Over the past few years machine learning has matured as an independent field as compared to a small area of artificial intelligence that it once was. Before the advent of AI and machine learning, the task of 'learning' was only restricted to a hierarchy of mammals supposed to possess intelligence. Undoubtedly, humans were at the root of this hierarchy. With technological advances, today, we fantasize about creating machines that can best at perform every all intelligent function that a human is capable of. There might be another hundred to two hundred more years before RoboCop becomes reality but we have made significant advances (albeit in small steps) to believe that it is not impossible. Since a long time (on a technology time line) we have tried to build machines that can recognize faces, handwriting and speech, respond to our questions, even react to our facial expressions, beat the world champion in a game of chess and can drive vehicles. These examples should not leave any doubt about the impact the field of *machine learning* has made.

Machine learning encompasses techniques where a machine acquires knowledge from its previous experience and gets better with more experience. These techniques try to emulate the way humans learn. Given a few observations we build a deduction (or a model). With additional observations we update the initial deductions. The algorithms within machine learning follow a similar iterative process. Machine learning is a very broad discipline that has been influenced by other disciplines such as artificial intelligence, statistics, philosophy, information theory and computational complexity. As stated above, learning is based on existence of past data. The learning task is broadly divided into two categories based on the information associated with the data supervised learning and unsupervised learning.

1.1 The Learning Spectrum

Supervised learning and unsupervised learning form the two extremes of the learning spectrum. Supervised learning, as the name suggests, has additional information (a tag/label for the data which is formally known as the *class* of

the data) regarding the data that guides the learning process. Whereas unsupervised learning lacks this information.
Supervised learning creates a function (also known as model or deduction) from training data (previous experience). The training data consists of pairs of input objects (typically vectors) and their classes. The learnt function can be a continuous value (called regression) , or can predict a class label of the input object (called classification). The task of the supervised learner is to predict the value of the function for any valid input object after having seen only a small number of training examples (i.e. pairs of input and target class). To achieve this, the learner has to generalize from the presented data to unseen situations in a "reasonable" way. In order to solve a given problem of supervised learning (e.g. learning to recognize handwriting) one has to consider various steps:

1. Determine the type of training examples. Decide what kind of data is to be used as an example. For instance, this might be a single handwritten character, a entire handwritten word, or a entire line of handwriting.

2. Gathering a training set. The training set needs to be characteristic of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.

3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the *curse of dimensionality*[1]; but should be large enough to accurately predict the output.

4. Determine the structure of the learned function and corresponding learning algorithm. For example, the you can choose to use neural networks or decision trees.

[1]Curse of dimensionality refers to the spareness of data points that results with the increase in dimensionality. This results in low precision of an estimator.

5. Run the learning algorithm on the gathered training set. Parameters of the learning algorithm may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation . After parameter adjustment and learning, the performance of the algorithm may be measured on a test set that is separate from the training set.

Based on the problem domain, characteristics of the data and constraints on the expected results different supervised learning approaches or algorithms could be applied. The following is a list of some of these approaches:

- Analytical learning
- Artificial neural networks
- Boosting
- Bayesian statistics
- Case-based reasoning
- Decision tree learning
- Inductive logic programming
- Gaussian process regression
- Learning automata theory
- probably approximately correct learning
- Support vector machines

These supervised learning approaches have been applied extensively in the fields of bio-informatics, handwriting recognition, information retrieval, object recognition in computer vision, spam detection, etc.
Unsupervised learning is a method of machine learning where a model is fit to the observations. It is distinguished from supervised learning by the fact that there is no a priori class information. In unsupervised learning, a data set of input objects is gathered. Unsupervised learning then typically involves finding

a "useful" representation of the data. By useful we could mean a representation for finding clusters or for dimensionality reduction or for finding hidden causes or sources of the data. Unsupervised learning is useful for data compression, outlier detection and even for supporting classification.

Between the two extremes of the spectrum lie various **semi-supervised learning** techniques. Semi-supervised because they use different combinations of labeled and unlabeled data. Some use positive labeled data with unlabeled data while some use a small set of labeled data along with a large set of unlabeled data.

1.2 Learning from Unlabeled and Labeled Data

At first glance, it might seem that nothing can be gained from unlabeled data. After all, an unlabeled data instance doesn't contain the most important piece of information its class. Here is an intuitive example of how unlabeled data might be useful. Suppose we are interested in recognizing web pages about academic courses. We are given just a few known course and non-course web pages, along with a large number of web pages that are unlabeled. By looking at just the labeled data we determine that pages containing the word *'homework'* tend to be about academic courses. If we use this fact to estimate the classification of the many unlabeled web pages, we might find that the word *'lecture'* occurs frequently in the unlabeled examples that are now believed to belong to the positive class. This co-occurrence of the words *'homework'* and *'lecture'* over the large set of unlabeled training data can provide useful information to construct a more accurate classifier that considers both *'homework'* and *'lecture'* as indicators of positive examples.

On a more formal note, what information does unlabeled data provide? By themselves they give us knowledge only of the distribution of examples in feature space - frequency of features over the unlabeled set. In the most general case, distributional knowledge will not provide helpful information to supervised learning. Zhang and Oles [1] provide a probabilistic analysis on the influence of unlabeled data for classification problems. They measure the value of unlabeled data based on its impact on the efficiency of parameter estimation.

Various approaches have been tried to exploit unlabeled data in order to

increase the accuracy of the classifier or to speed up the learning process. Two key approaches are as follows:

1. Train a classifier based on the labeled and unlabeled data. The labels for unlabeled data are imputed by certain means based on the current state of the classifier. The augmented data set is used to retrain the classifier.

2. Train a classifier based only on the labeled data. Then based on the current state of the classifier, one selects the "most significant" subset of the unlabeled data such that knowing the label of the selected data would maximally enhance the construction of the classifier. This technique is often called as *active learning.*

Nigam et. al. [2] employ a maximum likelihood (ML) estimation based approach (Expectation Maximization) that improved text classification accuracy as more unlabeled data was added. The ML technique is based on a generative model for document instances. The generative model encodes which words are more common in one class than another. Using this, it creates a document in a given class by randomly selecting words according to the class's word frequencies. This statistical models might not capture the complexity of human authoring process but these assumptions encode a relationship between the document distribution and the classification task that allow unlabeled data to be incorporated into learning. They use labeled and unlabeled documents to arrive at the most likely model parameter values. Their technique will be discussed further in context of the naïve Bayes model and also as a comparative technique. ML based techniques have been used by Lui et. al. [3] to classify documents using positive and unlabeled data. Apart from text classification, similar approaches have been successfully applied for face recognition and protein classification. Joachims [4] introduces *transductive support vector machines* that use a small set of labeled data to label a large unlabeled set with as little error. Semi-supervised SVMs (S^3VM) are introduced by Bennett [5] to solve the transduction problem estimating the class of each point in the unlabeled set.

The second approach for utilizing unlabeled data have been explored by Seung et. al [6] where they select an example from the unlabeled set based

on the *maximal disagreement* principle between a set of learners. McCallum and Nigam [7] improved the Query-by-Committee approach by Seung [6] by incorporating EM along with active learning. Their experiments show a vast reduction in the number of labeled instances required to achieve the same level of accuracy without their enhancements.

Other applications of unlabeled data with labeled data include building an ensemble of classifiers by Bennett et. al [8]. Another approach along the lines of using multiple learners was introduced by Blum [9]. In this approach Blum and Mitchell propose to use unlabeled data along with a natural split of the features in the data to boost accuracy of the resultant classifier. In this thesis, we look at enhancements to their technique, which they call **co-training**.

While unlabeled data has helped overcome the lack of labeled data in many applications, there have been instances where unlabeled data has deteriorated performance. Cozman et al. [10] conducted experiments on synthetic data aimed at understanding the value of unlabeled data. They reported that the classification accuracy could degrade more and more as more unlabeled data is added. Cozman et al. found that the reason for the degradation is the mismatch of the model assumption and the ground truth data distribution.

1.3 Text Categorization

The automated categorization (or classification) of texts into predefined categories has witnessed a booming interest in the last 10 years, due to the increased availability of documents in digital form and the ensuing need to organize them. In the research community the dominant approach to this problem is based on machine learning techniques: a general inductive process automatically builds a classifier by learning, from a set of pre-classified documents, the characteristics of the categories.

Text classification is of great practical importance today given the massive volume of online text available. In recent years there has been an explosion of electronic text from the World Wide Web, electronic mail, corporate databases, chat rooms, and digital libraries. One way of organizing this overwhelming amount of data is to classify it into descriptive or topical taxonomies. For

example, Yahoo maintains a large topic hierarchy of web pages. By automatically populating and maintaining these taxonomies, we can aid people in their search for knowledge and information. How are automatic text classifiers created? Early attempts were based on the manual construction of rule sets. Using this approach a person must compose a detailed set of rules for automatically specifying the class of a document. For example, one such rule might read "If the job posting contains the phrase 'expertise in Java' then the job category is computer programmer." Highly accurate text classifiers were built with this approach, but at significant cost. Constructing a complete rule set requires a lot of domain knowledge and a substantial amount of human time to tune the rules correctly. Such a manual approach is a huge impediment for customization. Imagine hand-crafting personalized email filtering rules for everyone who uses email. Since With few exceptions, this is an impractical approach to text classification.

A more efficient approach is to use supervised learning to construct a classifier. Here, we provide an algorithm with an example set of documents for each class, and allow it to find a representation or decision rule for classifying future documents. This approach also gives high-accuracy classifiers, and is significantly less expensive than manual construction because the algorithm automatically constructs the decision rule itself. Supervised text classification algorithms have been successfully used in a wide variety of practical domains. A few examples are:

1. Cataloging news articles citelewis[4] and web pages [11][12],
2. Learning the reading interests of users [13],
3. Sorting electronic mail [14].

However, the supervised learning approach is not as effortless as we might hope. One key difficulty with these algorithms is that they require a large, often prohibitive, number of labeled training examples to learn accurately. Labeling must typically be done by a person; this is a painfully time-consuming process. One would obviously prefer algorithms that can provide accurate classifications after hand labeling only a dozen articles, rather than thousands. This need

for large quantities of expensive labeled examples raises an important question: what other sources of information can reduce the need for labeled data? The task of learning text classifiers poses a set of challenges for machine learning:

1. *Large Input Space:* Since each keyword in the document (most of the times) serves as a single feature in text classification, the number of keywords is very large.

2. *Little Training Data:* For most learning algorithms, the required number of training examples to produce a sufficiently accurate classification rule scales with the dimensionality of of the input space. Since this condition cannot be met in the text domain, we can say that the amount of training data is scarce.

3. *Noise:* Text data is replete with incorrect spellings, typos and use of incorrect grammar. This qualifies as noise in machine learning terminology. Noise has the effect of deviating the accuracy of experiments.

4. *Complex Learning Tasks:* The concept to be learnt in text domain can be extremely complex such as learning the tone of a text message and finding the reading preference of a person. Since there is not formal model that encapsulates these tasks, the learner has to make do with an approximation.

5. *Computation Efficiency:* As mentioned earlier, the number of features in the text domain makes the task computationally very intensive. We need algorithms that can scale to such dimensions of the feature.

1.4 Objective

After having discussed the various components that influence the work in this thesis let us take a closer look at the motivation behind this thesis.

In general, unlabeled examples are much less expensive and easier to come by than labeled examples. This is particularly true for text classification tasks involving online data sources, such as web pages, email, and news stories, where huge amounts of unlabeled text are readily available. Collecting this text can frequently be done automatically, so it is feasible to quickly gather a large set

of unlabeled examples. If unlabeled data can be integrated into supervised learning then building text classification systems will be significantly faster and less expensive than before.

The above issues will be discussed again, in detail, in chapter 3.

1.5 Road map

The next chapter introduces the user to simple generative models, namely - Naive Bayes and Support Vector Machines used for text classification.

Chapter 3 introduces the reader to the co-training setting - the conditions for co-training and the algorithm. This chapter also introduces the contributions of the thesis towards enhancing the co-training setting.

Chapter 4 discusses the feature splitting techniques used for splitting the keyword feature space. It also discusses how the co-training conditions are preserved while performing these splits.

Chapter 5 talks about the experiments performed to show that co-training works better than other techniques that use unlabeled data. It also shows that co-training can be scaled to k-training. This chapter also shows that there are feature splitting techniques that capture the co-training conditions and result in improved accuracy for k-training.

Chapter 6 reiterates the conclusions discussed in this thesis and outlines possible applications of these conclusions.

Chapter 2
Models for Text Categorization

2.1 Introduction

Various classification models have been applied to text categorization but Naïve Bayes and Support Vector Machines have been the most popular because of their favorable properties. Naïve Bayes provides a convenient model with a strong foundation in statistical learning whereas SVMs provide a robust discriminative classifier which in insensitive to the dimensionality of the data. This chapter introduces these two classification models which have been used as base classifiers in the co-training setting. Simplicity of use and good success prompted the use of the naïve Bayes model. For the sake of comparison we conduct experiment SVMs also.

2.2 The Naïve Bayes Classification Model

The naïve Bayes model has had much success in text classification. Naïve Bayes has been used for spam filtering [14], authorship attribution, topic detection [15]. The naïve Bayes model has its root in probabilistic modeling which goes back to Maron's work [16] on automatic indexing of document. This section presents a probabilistic framework for characterizing the nature of documents and classifiers. The framework makes strong assumptions about how the data is generated. The following assumptions are made by the generative model and the naïve Bayes model is build in keeping with these assumptions:

1. the data is produced by a mixture model,

2. there is a one-to-one correspondence between mixture components and classes, and

3. the mixture components are *multinomial distributions* of individual words the words of a document are produced independently of each other given the class. According to McCallum and Nigam [17] a *multinomial model*[1]

[1]This model represents a document as a vector of binary attributes indicating which words occur and do not occur in the document. The number of times a word occurs in a document is not captured. When calculating the probability of a document, one multiplies the probability

has lower error rates as compared to a *multi-variate Bernoulli*[2] generative model.

From these assumptions a naïve Bayes classifier can be derived, by finding the most probable parameters for the model. Documents are generated by a mixture of multinomials model, where each mixture component corresponds to a class. Let there be $\mid C \mid$ classes and a vocabulary of size $\mid V \mid$; each document d has $\mid d \mid$ words in it. How do we create a document using this generative model? First, we roll a biased $\mid C \mid$-sided die to determine the class of our document. Then, we pick up the biased $\mid V \mid$-sided die that corresponds to the chosen class. We roll this die $\mid d \mid$ times, and write down the indicated words. These words form the generated document. Formally, every document is generated according to a probability distribution defined by the parameters for the mixture model, denoted θ. The probability distribution consists of a mixture of components $c_j \in \mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}$. Each component is parameterized by a disjoint subset of θ. A document, d_i, is created by first selecting a mixture component according to the mixture weights (or class probabilities), $P(c_j \mid \theta)$, then having this selected mixture component generate a document according to its own parameters, with distribution $P(d_i \mid c_j; \theta)$ [3]. Thus, we can characterize the likelihood of document d_i with a sum of total probability over all mixture components:

$$P(d_i \mid \theta) = \sum_{j=1}^{|\mathcal{C}|} P(c_j \mid \theta) P(d_i \mid c_j; \theta) \tag{1}$$

Each document has a class label. We assume that there is a one-to-one correspondence between mixture model components and classes, and thus use c_j to indicate the j^{th} mixture component, as well as the j^{th} class. The class label for a particular document d_i is written y_i. If document d_i was generated by mixture component c_j we say $y_i = c_j$. The class label may or may not be known for a given document. A document, d_i, is considered to be an ordered

of all the attribute values, including the probability of non-occurrence for words that do not occur in the document.

[2]This model represents a document by the set of word occurrences from the document. As in the multi-variate Bernoulli model, the order of the words is lost, however, the number of occurrences of each word in the document is captured. When calculating the probability of a document, one multiplies the probability of the words that occur.

[3]Standard notational shorthand for random variables is used, whereby $P(X = x_i \mid Y = y_j)$ is written $P(x_i \mid y_j)$ for random variables X and Y taking on values x_i and y_j.

list of word events, $\langle w_{d_i,1}, w_{d_i,2} \ldots \rangle$. We write $w_{d_i,k}$ for the word w_t in position k of document d_i, where w_t is a word in the vocabulary $V = \langle w_1, w_2, \ldots, w_{|V|} \rangle$. When a document is to be generated by a particular mixture component a document length, $\mid d_i \mid$, is first chosen independently of the component. (Note that this assumes that document length is independent of class.). Then, the selected mixture component generates a word sequence of the specified length. We assume it generates each word independently of the length. Thus, we can expand the second term from Equation 1, and express the probability of a document given a mixture component in terms of its constituent features: the document length and the words in the document. Note that, in this general setting, the probability of a word event must be conditioned on all the words that precede it.

$$\begin{aligned} P(d_i \mid c_j; \theta) &= P(\langle w_{d_i,1}, \ldots w_{d_i,|d_i|} \rangle \mid c_j; \theta) \\ &= P(\mid d_i \mid) \prod_{k=1}^{|d_i|} P(w_{d_i,k} \mid c_j; \theta; w_{d_i,q}, q < k) \end{aligned}$$

Next we make the standard naïve Bayes assumption: that the words of a document are generated independently of context, that is, independently of the other words in the same document given the class label. We further assume that the probability of a word is independent of its position within the document; thus, for example, the probability of seeing the word 'help' in the first position of a document is the same as seeing it in any other position. We can express these assumptions as:

$$P(w_{d_i,k} \mid c_j; \theta; wi_{d_i,q}, q < k) = P(w_{d_i,k} \mid c_j; \theta)$$

Combining these last two equations gives the naive Bayes expression for the probability of a document given its class:

$$P(d_i \mid c_j; \theta) = P(\mid d_i \mid) \prod_{k=1}^{|d_i|} P(w_{d_i,k} \mid c_j; \theta)$$

Thus the parameters of an individual mixture component define a multinomial distribution over words, i.e. the collection of word probabilities, each written θ_{w_t,c_j}, such that $\theta_{w_t,c_j} \equiv P(w_t \mid c_j; \theta)$, where $t = \{1, \ldots, \mid V \mid\}$ and $\sum_t P(w_t \mid c_j; \theta) = 1$. Since we assume that for all classes, document length is identically

distributed, it does not need to be parameterized for classification. The only other parameters of the model are the mixture weights (class probabilities), written θ_{cj} , which indicate the probabilities of selecting the different mixture components. Thus the complete collection of model parameters, θ, defines a set of multinomials and class probabilities:
$\theta = \{\theta_{w_t|c_j} : w_t \in V, c_j \in C; \theta cj : c_j \in C\}$

2.3 Learning with Unlabeled Data

Chapter 3
Co-training Setting

3.1 Introduction

The applicability of the co-training setting can be explained by the following plausible scenario. You have three thousand e-mails in your Inbox when you arrive from a long vacation. Eager to get on with work as soon as possible, you don't want to spend time wading through these messages trying to first, separate solicited messages from the unsolicited (spam) messages and then categorize the non-spam messages into specific folders. At this moment, how much yearn for a mail agent that would work like magic and sort all your e-mails into the right categories. This might be a little idealistic, but you will definitely settle for a little less your mail client asks you to categorize just a few e-mails and guarantees to sort the remaining e-mails with a very high accuracy. With three thousand messages in your Inbox, this feels like a very good option in terms of the time saving. Like any situation where you can think of applying semi-supervised learning the above scenario also has a some labeled data and lots of unlabeled data. This leaves no doubt about co-training being a semi-supervised learning technique.

What other characteristics of data (an email message for our example) is helpful in applying the co-training setting. This might not be very obvious but an email message has two distinct parts (body and subject), which can be assumed to be mutually independent. The following sections will outline how this property of the data is helpful for the co-training setting. Other scenarios where the co-training algorithm can be applied include classifying network intrusion attacks, identifying various types of diseases/tumors/bacterias.

The following sections will discuss the co-training algorithm, the conditions that need to be satisfied to apply the co-training setting. By the end of the chapter, we will introduce the issues that this thesis work aims at addressing.

3.2 What is Co-training

Co-training is a semi-supervised learning technique of building a classifier using labeled data along with unlabeled data. Since labeled data is hard to

obtain, either due to scarcity or lack of labeling effort, co-training leverages the information in the unlabeled data to augment the classifier performance. The unlabeled data is used to boost the performance of a weak classifier build using only the labeled data. The concept of co-training was introduced by Blum and Mitchell [9] wherein they describe a model of learning with unlabeled data especially when an example can be expressed by two distinct views. Blum and Mitchell [9] applied co-training to solve the problem of auto-classifying web pages based on the text in those web pages and the text on links (in other pages) that referenced those web pages. Each web page could be thus be described in two possible ways first, in terms of the words in the web page and second, in terms of the words contained in links (in other web pages) that referenced that web page. In other words, there are two feature sets that can describe a document. The terms 'feature sets' and 'views' would be used interchangeably during the course of this document. Such a dual-view characteristic can be observed in a number of datasets. An email message for example can be described in terms of the keywords in the subject and keywords in the body of the message. Any image can be described in terms of features in RGB color domain and features in HSI color domain. Such a duality can be obtained even while describing the characteristic of a person. Every person can be described by either his physical characteristics (height, weight, color of skin, etc.) or his social characteristics (employer, education, plays sports, salary, etc.).

3.2.1 The Algorithm

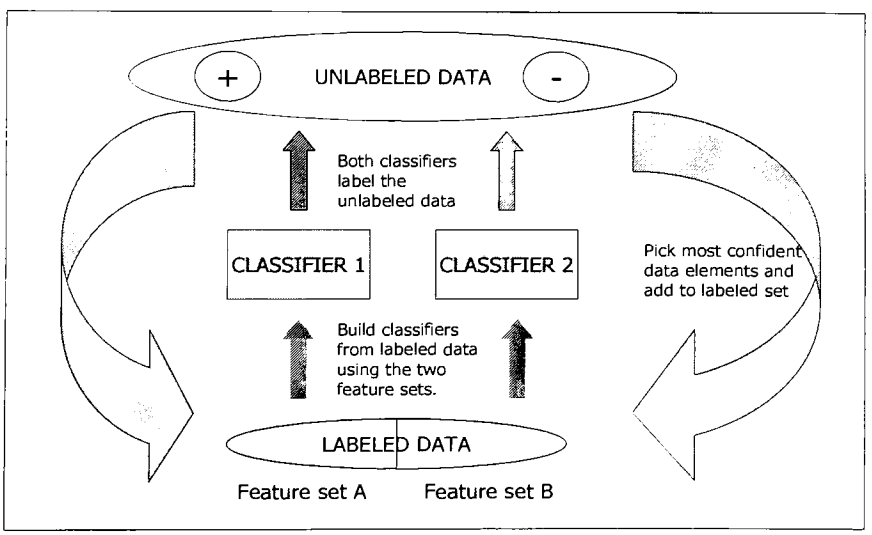
The availability of large number of unlabeled documents and the existence of two views to describe each example are key for the co-training setting. The samples are drawn from an instance space $X = (X_1, X_2)$, where X_1 and X_2 are from different observations. If f is the target function over D, then for any example $x = (x_1, x_2)$ we should have $f(x_1, x_2) = f_1(x_1) = f_2(x_2)$ where f_1 and f_2 are the target functions over X_1 and X_2 respectively.

The working of the co-training algorithm is depicted in figure 1 and the algorithm is described in table 1. From the labeled documents, two classifiers are trained one for each view of the document. These two models (f_1^0 and f_2^0) represent the initial weak classifiers. 'Weak' since they are trained from a small set

of labeled examples. The final objective is to strengthen (*bootstrap*) these weak classifiers by reducing the disagreement between them. Both f_1^0 and f_2^0 label all the unlabeled documents after which each of them picks the most confidently labeled examples (n with negative label and p with positive label) and adds them to the labeled set of documents. In the subsequent iteration, new classifiers f_1^1 and f_2^1 are trained from the augmented labeled set. This process of adding most confidently labeled documents to the labeled set and retraining continues till all the unlabeled documents are exhausted. The algorithm outputs the final pair of classifiers f_1^T and f_2^T, where T is the total number of iterations needed to exhaust all unlabeled documents.

During each iteration the two classifiers try to reduce the degree of disagreement between them. The disagreement can be measured in terms of the unlabeled samples for which their labeling disagrees. It has been shown in [18] that minimizing the disagreement between two individual models could lead to the improvement of the classification accuracy of individual models.

Figure 1: Co-training Algorithm



Datasets whose features naturally partition into two sets, and algorithms that use this division fall under the co-training setting.

The objective of the co-training algorithm can be stated by the following

Table 1: The Co-training Algorithm

Given:

- a set L of labeled training examples.
- a set U of unlabeled training examples.

Create a pool U' of examples by choosing u examples at random from U.
Loop for k iterations:

Use L to train a classifier h_1 that considers only the x_1 portion of x.

Use L to train a classifier h_2 that considers only the x_2 portion of x.

Allow h_1 to label p positive and n negative examples from U'.

Allow h_2 to label p positive and n negative examples from U'.

Add these self-labeled examples to L.

Randomly choose $2p + 2n$ examples from U to replenish U'.

theorem in [9]

Theorem 3.2.1 *Given a weak hypothesis $h_1 \in \mathcal{H}_1$, the true labeling function f is learnable in the co-training model if f_2 is learnable in the PAC model with classification noise and view-independence.*

3.2.2 Co-training Assumptions

The following two assumptions regarding the data are sufficient for the co-training algorithm:

1. The feature sets or views should satisfy the *conditional independence* criterion. V_1 is *view-independent* of V_2 if and only if:

$$\forall a \in V_1, b \in V_2, Pr_{\mathcal{D}}[a \mid b] = Pr_{\mathcal{D}}[a]$$

   This property is also known as the *conditional independence* property. This property ensures that whenever an unlabeled example that is labeled by the first classifier is added to the labeled set, it boosts the second classifier. The following example provides an intuitive feel for this property.

The necessity of this condition can be explained in the next section which discusses the relation of co-training to PAC-style theoretical framework.

2. V_1 and V_2 are *view-compatible* if and only if:

$$\forall a \in V_1, b \in V_2, Pr_{\mathcal{D}}[a \mid b] \neq 0 \longrightarrow f_1(a) = f_2(b)$$

This property is also known as the *redundancy property*. The existence of two views that have similar classification strength rendering the other view as redundant leads to the name of this property.

Conditional Independence

Conditional independence

$$\begin{aligned} P(Image \mid Identity, Voice) &= P(Image \mid Identity) \\ P(Voice \mid Identity, Image) &= P(Voice \mid Identity) \end{aligned}$$

If conditional independence exists, an image instance is paired with any voice instance according to the distribution of voice instances, regardless of what the image is.

In order to justify a PAC-style framework for co-training [9] introduces the concept of *compatibility* between the distribution $\mathcal{D}$ and the target function f. A target function $f = (f_1, f_2) \in C_1 \times C_2$ is said to be *compatible* with a distribution if, whenever $f_1(x_1) \neq f_2(x_2)$, then $\mathcal{D}$ assigns a probability zero to the sample (x_1, x_2). It is fairly safe to say that the number of concepts that might be compatible with the distribution will at the most be equal to the total number of concepts, which in our example is $\mid C_1 \mid \times \mid C_2 \mid$. For most practical purposes the number of compatible concepts is much less than the total concepts. The role of the unlabeled samples is to help narrow down the number of competing concepts. Without the unlabeled examples, we would have needed more labeled data to eliminate weak concepts. Since the unlabeled data is also assumed to adhere to the compatibility criterion, every time an unlabeled data is correctly classified by the weak classifier, it adds information that would have otherwise needed additional number of labeled data. This can be explained with the help of an example. Suppose an unlabeled email subject has the keyword "free"

and the body of the email has the keywords "lose weight" Also suppose that the weak classifier built from keywords in the subject of labeled email messages labels an email as spam (with high probability) if it encounters the keyword "free" When the unlabeled email is classified as spam and used as training data for the other classifier it adds the information that the keywords "lose weight" in the body are indicative of a spam email.

The following problem domains have applied co-training to exploit a natural split of the data into two views:

1. Named entity classification (spelling vs. context) [19]
2. Web page classification [9]
3. Word sense disambiguation [20].
4. Email classification [21].

In 'real world' it is hard to find datasets which have a natural split that satisfies the above two conditions required to apply the co-training algorithm. The question arises whether the co-training algorithms can be applied to datasets that do not possess such as natural split. In [22], Nigam and Ghani have explored an alternate solution for datasets with lack of natural splits. Their experiments on an engineered dataset shows that features splits can be constructed for datasets that do not have natural splits.

In [22] the authors show that when learning from labeled and unlabeled algorithms that explicitly leverage a natural split of the features outperform those that do not.

3.3 PAC Learning

Computational learning is concerned with learning algorithms which are efficient and have provable error rate bounds. One of the most popular models of learning satisfying these conditions is the probably approximately correct (PAC) model of learning introduced by Valiant [23]. Within the PAC model, the learning algorithm must infer a concept from a sequence of labeled examples. A concept is simply a rule which partitions objects from a domain into one of two categories: positive examples and negative examples. In an instance of PAC

learning, a learner is given the task of determining a close approximation of an unknown, [0,1]-valued target function from labeled examples of that function. The learner's goal is to output, with probability at least 1-d, an hypothesis h whose error rate is at most e, for the given accuracy parameter e and confidence parameter d. A learning algorithm is said to be polynomially efficient if its running time is polynomial in $1/\epsilon$, $1/\delta$ and n. The PAC learning model is often referred to as the strong learning model since the learning algorithm may be required to output an arbitrarily accurate hypothesis.

Let X refer to the set of all possible instances over which target functions may be defined. Let C refer to some set of target concepts that our learner might be called upon to learn. Each target concept c in C corresponds to some subset of X, or equivalently to some boolean-valued function $c : X \rightarrow 0, 1$. We assume instances are generated at random from X according to some probability distribution D. All that we require of D is that it be stationary; that is, that the distribution not change over time. Training examples are generated by drawing an instance x at random according to D, then presenting x along with its target value $c(x)$, to the learner. The learner L considers some set H of possible hypotheses when attempting to learn the target concept. For example, H might be the set of all hypotheses describable by conjunctions of some attributes. After observing a sequence of training examples of the target concept c, L must output some hypothesis h from H, which is its estimate of c. To be fair, we evaluate the success of L by the performance of h over new instances drawn randomly from X according to the same probability distribution D. Within this setting, we are interested in characterizing the performance of various learners L using various hypothesis spaces H, when learning individual target concepts drawn from various classes C. Since we are interested in how closely the learner's output hypothesis h approximates the actual target concept c, let us begin by defining the *true error* of a hypothesis h with respect to target concept c and instance distribution D. Informally, the true error of h is just the error rate we expect when applying h to future instances drawn according to the probability distribution D. The true error rate (denoted $error_D(h)$) of the hypothesis h with respect to the target concept c and distribution D is the probability that

h will misclassify an instance drawn at random according to D.

$$error_D(h) = Pr_{x \epsilon D}[c(x) \neq h(x)]$$

, where the probability is taken over the instance distribution D. The figure shows a Venn diagram representation of the overlap between the target concept and the hypothesis h picked by the learner L. The shaded area shown the regions of dissimilarity between the hypothesis and the target concept. Since the error rate is over the entire distribution, it depends on the probability of occurrence of the instances that fall under the dissimilar region between c and h. For example, if there were a large number of instances that were labeled dissimilarly by h and c, but their probability of occurrence was very low then the error rate would be much less than if the instance distribution was uniform or biased towards these instances. On the other hand, if the probability for the instances in the region of dissimilarity is high then the error rate would be high. Thus the error rate depends on the probability function for the instance distribution. Unfortunately, the learner is unable to gauge the correctness of the hypothesis it is choosing since it does not have access to new instances. The learner can only test its hypothesis on labeled data (also known as *training data*). The error of the hypothesis on the training data is defined as the *training error*. The objective of PAC learning is to characterize classes of target concepts that can be reliably learned from a reasonable number of randomly drawn training examples. The primary property that PAC learning intends to capture is the number of training examples needed to generate a hypothesis that has zero error rate. Unfortunately, this is impossible to achieve, primarily due to the following two reasons: 1. There is a possibility that more than one hypotheses confirm to the training examples seen so far. So the learner does not know which one to pick. 2. There is always a chance that the examples produced by the distribution are misleading. In keeping with the above two conditions, we need to relax our demands on the PAC setting. Firstly, we do not expect the learner to output a hypothesis that has zero error rate. We need to be satisfied as long as the hypothesis has an error rate less than ϵ. Since there might be more than one hypotheses that comply with the training data, we cannot be sure that the learner has picked the correct hypothesis. Hence, we

have to allow for a possibility of error in choosing the hypothesis. This error can be represented by δ. So the learner has to pick a correct hypothesis with probability at least $(1-\delta)$. Since the learner hopefully (with probability $(1-\delta)$) picks an *approximately correct* hypothesis this setting is called the Probably Approximately Correct learning setting. [24] formally defines PAC learnability as: "Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is **PAC-learnable** by L using H if for all $c \epsilon C$, distribution D over X, ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1 - \delta)$ output a hypothesis $h \epsilon H$ such that $error_D(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n, and $size(c)$. This definition my appear to focus on the computational resources required for learning but practically we are more concerned with the number of training examples required. The computation time can be related to the number of training examples if we can show that the learner requires a bounded time for processing each training example. The growth in the number of required training examples with problem size is called the *sample complexity.* A learner is called a *consistent learner* if it outputs a hypothesis that perfectly fits the training data (whenever possible). A *version space* is defined as the set of all hypotheses that correctly classify the training examples. Formally:

$$VS_{H,D} = \{h \epsilon H \mid (\forall \langle x, c(x) \rangle \epsilon D)(h(x) = c(x))\}.$$

A bound on the number of training examples required for learning a concept class can be shown for a consistent learner. Since a consistent learner always outputs a hypothesis that belongs to the version space, to prove the bound for a consistent learner we need to find the minimum number of examples that would always result in a hypothesis from the version space. Or in other words we need to make sure that the version space contains hypothesis that perfectly fit the training data. According to [24] a version space $V_{H,D}$ is **ϵ-exhausted** with respect to the concept class c and the instance distribution D, if every hypothesis h in $V_{H,D}$ has error less than ϵ with respect to c and D.

$$(\forall h \epsilon VS_{H,D})\, error_D(h) < \epsilon$$

[25] provides a probability bound for the ϵ-exhaustibility of a version space. It states that the probability that a version space $VS_{H,D}$ is not ϵ-exhausted (with

respect to the c) is less than or equal to

$$\mid H \mid e^{-\epsilon m}$$

where m is the number of training examples. The above equation bounds the number of training examples m needed to eliminate all hypotheses having true error greater than ϵ. This inequality can be used to get a bound on number of examples needed to get a version space that is with probability less than δ not ϵ-exhausted. This statement can be represented as:

$$\mid H \mid e^{-\epsilon m} \leq \delta$$

which can be solved to get

$$m \geq \frac{1}{\epsilon}(\ln \mid H \mid + \ln(1/\delta))$$

The above bound on m depends on the number of hypotheses in the hypothesis space. Hence this bound is applicable only for cases where $\mid H \mid$ is finite. For cases where $\mid H \mid$ is infinite we define another measure of complexity.

3.4 Questions Asked In This Thesis?

The co-training setting seems to work well when the assumptions are satisfied. Nigam [22] showed that two views can be constructed if they are not present naturally. This thesis aims at extending the work presented in [22]. This thesis asks and answers the following questions:

1. If a natural split is not present how well can a split be constructed? The thesis looks at various feature splitting techniques that can preserve the above conditions. The experiments use randomly constructed views as a benchmark. Initial experiments have shown that at times constructed splits work better even on datasets that do have a natural split.

2. Till now co-training has focused on learning two models over the sample. Is it possible to scale this up to k models? Is there a sense of optimal number of classifiers to be used? Can feature splitting techniques provide k partitions of the features that could be used with these k classifiers?

3. How sensitive is co-training to the underlying classifier. What happens when different classifiers are used? In this thesis we experiment with the Naïve Bayes classifier and the Support Vector Machines based classifiers.

Chapter 4
Enhancing the Co-training Setting

4.1 Introduction

As discussed in the previous chapter co-training has been used for dataset that have a natural split of the features. Looking at the favorable results obtained by formulating a problem with this setting we are intrigued by the possibility of applying this setting to problems that do not display a natural split in its features. So we investigate two questions:

1. Can we construct a split in the features that could give us the same improvement as co-training on a natural split? In addition, such a pseudo split should fulfil all the conditions that are required for the co-training setting?

2. If we can construct a single split, could this idea be extended to multiple splits? That is, could we split the features into multiple subsets such that a classifier per subset results in a better overall classifier.

The following sections discuss above mentioned issues in detail and elaborate on the approach taken in investigating those issues.

4.2 Co-training to k-training

Blum and Mitchell [9] explained the co-training setting with the help of a weighted bipartite graph representation of the data distribution. The nodes on the left hand side of the graph represent one of the views of the data that is generated from the instance space X_1. Each node on the left hand side is a possible set of features (or keywords in our case) that has a non-zero likelihood of being generated by the instance space. Similarly, the nodes on the right hand side of the graph represents all possible candidates for the other view of the data generated from the instance space X_2. Again, each node represents a combination of features from the instance space X_2 that have a non-zero probability of occurring together. An instance generated by the distribution over $X = X_1 \times X_2$ is represented by an edge between two nodes on either side of the graph. The weight on the edge is the probability of occurrence of the

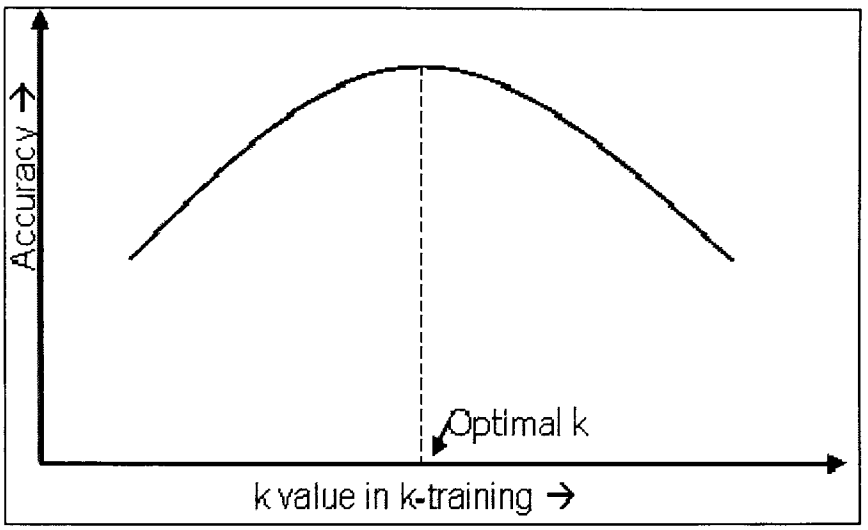
pair in the distribution. The compatibility condition stated in chapter 3 can be represented by those partitions that do not have a cross edge between them. By the co-training assumption, for labeled instances, both nodes corresponding to that instance (and which are also connected by an edge) edge have the same label. Moreover, each node in a compatible partition will have the same label. The two views are assumed to be conditionally independent. When an unlabeled instance (x_1, x_2) is added to the graph such that x_1 belongs to a labeled partition of the graph. This results in x_2 being added to the same label. Since x_2 is used by another classifier, this information that x_2 is labeled helps improve the classifier. Obviously, the assumption is that x_2 is given the correct label. In case x_2 is given an incorrect label the second classifier deteriorates, resulting is an inaccurate co-trained classifier.

This concept of using two classifiers, each using a separate partition of the features to train, can be extended to more than two partitions. Hence the name *k-training*. Now, instead of X_1 and X_2 there are views X_1 to X_n. As before, the weight on the edges is the probability of occurrence of the pair in the distribution. Each instance is represented by a set $(x_1, x_2, \ldots, x_n)$. We hypothesize that, for a dataset, with an increase in the number of views (the value of k) the performance of a k-trained classifier improves till it reaches a peak after which the performance deteriorates. The peak represents the point at which the feature sets are maximally independent. With the increase in the number of views the feature set gets divided into smaller partitions which violate the conditional independence criterion resulting in an increase in the classification noise. As each classifier gets injected with more and more classification noise at each iteration of the k-training algorithm the performance deteriorates. The hypothesized performance of the k-trained classifier is shown in figure 2.

4.3 Constructing Feature Partitions

Nigam and Ghani [22] toyed with the idea of constructing pseudo partitions of the feature set for co-training. They showed that a constructed feature split can perform as well as a natural feature split. They constructed a dataset by merging articles from two newsgroups in the 20 Newsgroups dataset which discussed disconnected issues for example baseball and mideast political issues.

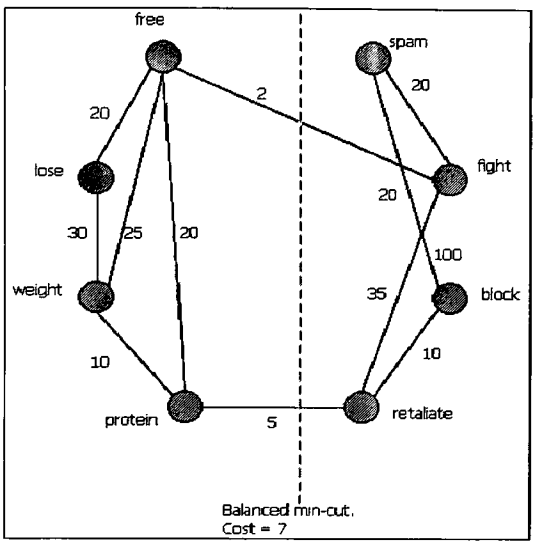
Figure 2: K-training Versus Accuracy - Hypothesis



On this dataset, Nigam and Ghani performed a random split of the feature set and used each split to train a classifier. Their results did not show any definite improvement. With this background, in this thesis we aim at going beyond a random split to obtain partitions that preserve the co-training assumptions hence helping in constructing a better classifier.

Following the idea of representing the feature splitting problem as a graph partitioning (GP) problem, we map each feature to a node of a graph. To formulate this problem the weights on the edges should involve a condition that we intend to minimize (or maximize). For the co-training criterion we need to find partitions that are conditionally independent. This means that we need to capture the notion of conditional independence in the weights on the edges such that a partition of the vertices involves cutting edges that maximize the independence criterion. The figure 3 intuitively conveys the idea. We use **mutual information** as the weight on the edges. This is discussed further in the following subsection.

Figure 3: Balanced Mincut



4.3.1 Mutual Information

Mutual information, or simply information, was introduced by Shannon in his landmark 1948 paper "A Mathematical Theory of Communication" Mutual information is an information theoretic measure of correlation between two random variables. Let us go through an informal example for understanding mutual information before we get into the math. Consider some example data (taken from a 20 million word corpus) for the word "united". It co-occurs with many words, among which are "states", "stand" and "airways". The observable facts are that "united" has an overall corpus freq of 2579 (let's refer to this as $f(united) = 2579$) and also $f(airways) = 5237$, $f(states) = 1019262$ and $f(stand) = 51$. We also observe the number of times these words co-occurred with "united" (for shorthand $j(states) = 1583$ to mean that "states" occurred with "unites" 1583 times: this is the "joint" frequency). So $j(states) = 1583$, $j(airways) = 297$, $j(stand) = 51$. Now if we were to rank similarity of these words to "united" by raw frequency of co-occurrence we would order them according to $j(x)$, as above. Of course, a full similarity listing of "united" in this form would have many other words with intermediate frequencies – we are just focussing on these three words for the moment. But the ordering show

above does not tell us anything much about the strength of association between "united" and these other words: it is simply a reflection of the basic overall frequency of the these words (i.e. "states" is much more frequent than "airways" which is much more frequent than "stand"). We just showed that in the $f(x)$ list! This is true in general: ordering similarity by $j(x)$ simply places words like "the", "a", "of", "to" at the top of every similarity list. We would like to know to what extent does the word "united" influence its lexical environment by selecting particular words with which it will co-occur? We can compare the relative frequencies of what we observed with what we would expect. The expected frequency is under the assumption that word "united" has no effect whatsoever on its neighboring environment and the frequencies of words surrounding "united" will be exactly the same as they would be if "united" were present or not.

$$\begin{aligned} E[j(states)] &= (f(united) \times span) \times relative\_freq(states) \\ &= (2579 \times 8) \times \frac{1}{20} = 1031 \end{aligned}$$

So the expected value of $j(states)$ is 1031. We actually observed $j(states)$ to be 1583, which is rather higher, and we could simply express the difference as ratio of observed to expected joint frequency thus. This is the **Mutual Information** score and it expresses the extent to which observed frequency of co-occurrence differs from expected. Of course, big differences indicate massive divergence and indicate that "united" is exerting a strong influence over its neighboring keywords.

This example gives an intuitive feel of what mutual information. In the example above, let X and Y be random variables corresponding to the number of occurrences of keywords "united" and "states". In information theoretic terms mutual information denotes the information one random variable tells about another one. We need to go through some definitions before we can formally define mutual information.

The basic concept of **entropy** in information theory has to do with how much randomness is in a signal or in a random event. As an example consider some English text, encoded as a string of letters, spaces and punctuation (so our signal is a string of characters). Since some characters are not very likely

(e.g. 'z') while others are very common (e.g. 'e') the string of characters is not really as random as it might be. Entropy is the complement of information. Higher the entropy lesser the information and lesser the entropy more certain the information.

Definition 4.3.1 *Shannon defines entropy in terms of a discrete random event X, with possible states $1 \ldots n$ as:*

$$H(X) = -\sum_{i=1}^{n} p(i) log_2 p(i)$$

Suppose there is a random variable with true distribution p. Then we could represent that random variable with a code that has average length $H(p)$[1] However, due to incomplete information we do not know p; instead we assume that the distribution of the random variable is q. Then the code would need more bits to represent the random variable. The difference in the number of bits is denoted as $D(p\|q)$ and is known as the **relative entropy**.

Definition 4.3.2 *The relative entropy or Kullback-Leibler distance between two probability mass functions $p(x)$ and $q(x)$ is defined as*

$$D(p\|q) = \sum_{x \in \mathcal{X}} p(x) log \frac{p(x)}{q(x)}$$

Definition 4.3.3 *Let the joint distribution between the random variables X and Y be $p(X, Y)$ and their marginal distributions be $p(X)$ and $p(Y)$. The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution:*

$$I(X;Y) = D(p(x,y)\|p(x)p(y)) = \sum_{x} \sum_{y} p(x,y) log \frac{p(x,y)}{p(x)p(y)}$$

Note that when X and Y are independent, $p(x, y) = p(x)p(y)$, so $I(X;Y) = 0$. This makes sense since independence between the two variables inhibits either of them to provide any information about the other. From the equation above it can be seen that mutual information is a symmetric measure that is

Theorem 4.3.1 *I(X;Y) = I(Y;X)*

[1]

Mutual information $I(X;Y)$ which is the information Y tells us about X, can also be interpreted as the reduction in uncertainty about X due to the knowledge of Y. This notion is captured in the following theorem.

Theorem 4.3.2

$$I(X;Y) = H(X) - H(X|Y)$$

and similarly

$$I(Y;X) = H(Y) - H(Y|X)$$

$$H(X|Y) = \sum_{Y} P(Y) \sum_{X} P(X|Y) log P(X|Y)$$

After the math we are ready to apply it to the problem at hand. In our case, the X and Y random variables represent the sets of keywords that would be partitioned. Computing $H(X|Y)$ would be computationally intensive since we will have to consider each permutation of keywords in X with each permutation of keywords in Y To approximate this we use *pointwise mutual information.* Pointwise mutual information is the mutual information between a pair of keywords.

$$I(x;y) = log \frac{p(x,y)}{p(x)p(y)}$$

So the mutual information between X and Y is approximated by the pointwise mutual information between each pair of keyword in X and Y.

$$I(X;Y) \approx \sum_{x \in X} \sum_{y \in Y} log \frac{p(x,y)}{p(x)p(y)}$$

The co-training setting requires class conditional independence between the feature sets, which in keeping with our previous discussion can be represented by *conditional mutual information.* Conditional mutual information is given by

$$I(X;Y|Z) = H(X|Z) - H(X|(Z,Y))$$

This value quantifies how much information is shared between X and Y, given the value of Z. Another way to see it, as it it decomposed above, is as the difference between the information required to describe X given Z, and the information to describe X given both Z and Y. If Y and Z carry the same

information about X, the two terms on the right are equal, and the conditional mutual information is zero. On the contrary if both Y and Z bring information, and if those information are complementary, the difference is large.

Hence the final weight on an edge of the graph connecting keywords x and y is given by the following expression[2]:

$$I(x;y|z) = \sum_{z \in Z} \sum_{y \in Y} \sum_{x \in X} P(x,y,z) \frac{P(x,y|z)}{P(x|z)P(y|z)}$$

Other measures such as log likelihood ratio, t-test or χ^2 test could also be used to measure the similarity between two keywords.

The next chapter discusses the graph partitioning algorithms we considered and the one that was finally used. for graph partitioning.

[2]The derivation of this formulae is shown in the appendix.

Chapter 5
Feature Partitioning Techniques

5.1 Introduction

Feature partitioning techniques encompass methods that divide the features into subgroups based on a given distance measure. Feature partitioning is closely related to feature selection but is not the same. Feature partitioning can be considered as a form of clustering wherein a set of features is divided into subsets. Each member of a subset is close (based on the distance measure used) to all the members of the same subset as compared to members of other subsets. Feature partitioning might seem to be closely related to feature selection but it differs in two main aspects. In feature selection, the goal is to pick the most relevant subset of features from a larger set whereas in feature partitioning the entire set of features is retained. Secondly, feature selection is closely related to ranking whereas feature partitioning is closely related to clustering (as mentioned above).

5.2 Feature Partitioning

Ample work has been done in the field of feature partitioning, which is also known as grouping features or feature clustering. In [26], the authors have addresses grouping attributes as well as the data points simultaneously. They use information theoretic measures (namely mutual information) to formulate the problem as an optimization problem. The iterative maximization of the mutual information leads to better clusters both in the feature space and in the data point space. Gautam Das et al. [27] use a set of other attributes in order to find the correlation between two attributes. The paper uses "external probes" (other attributes) rather than internal measures (defined purely in terms of the values in the attributes) to find relation between the features. Association rules [28] also capture the relation between attributes. Similarly, feature clustering algorithms have also been proposed in [29],[30]

5.2.1 Feature Partitioning under the Co-training Setting

For the co-training setting it is important that the two feature sets be independent of each other. This problem can be modeled as that of obtaining a minimum cost bisection of a graph. Each feature is represented as a vertex in the graph. Two vertices are connected by an edge if they share some information. For example, keywords "weight" and "loss" are considered to be related if they occur together. The presence of the first keyword affects the probability of occurrence of the second keyword. The weight on the edge is the conditional mutual information [31] between the keywords corresponding to the vertices. Now, we have a graph that represents the features and the correlation between them. In order to obtain two equally sized sets (bisection) that are independent, we need a cut of the graph that minimizes the inter-set correlation and maximizes the intra-cut correlation. In graph theory, this is known as minimum cost bisection problem. This problem is known to be NP-hard [32]. The best we can do is to find an approximate algorithm for finding the minimum cost bisection.

5.3 Minimum Cost Graph Bisection

A bisection of a graph $G = (V, E)$ with an even number of vertices is a pair of disjoint subsets $A, B \subset V$ of equal size. The cost of a bisection is the number of edges $c = (a, b) \in E$ such that $a \in A$ and $b \in B$. The problem of Graph Bisection takes as input a graph G with an even number of vertices, and returns a bisection of minimum cost.

This problem arises frequently in real world applications, such as parallel scientific computing, VLSI design, and task scheduling. For example, in scientific computing, it is common to use parallel computers to perform sparse matrix-vector multiplication. Typically each processor owns some fraction of the rows of the matrix, and is responsible for computing only those components of the result corresponding to rows it owns. In order to compute the result, however, it must have a valid entry in any component of the vector for which there is a nonzero entry in the corresponding column of the matrix in any row it owns. Thus the amount of communication which is necessary to perform a matrix-vector multiplication in parallel depends on how effectively the rows of

the matrix are distributed to the processors. If there are two processors, it is desirable to split the number of rows that each processor owns roughly in half (for load balancing), and to assign rows so that the number of nonzero matrix entries a_{ij} with i owned by one processor and j owned by the other is minimized (to minimize communication). If there are more than two processors, one typically uses graph bisection recursively until a good assignment of rows to processors is found.

5.3.1 Minimum Bisection is NP-Complete

We show that the minimum bisection problem and the problem of approximating it to within a constant factor are NP-complete.

- **MIN-CUT $\in$ NP**

The decision problem for minimum bisection, which asks for a given graph whether it has a bisection of less than a given width, is obviously in NP since given a bisection we can verify quickly its width and the fact that it is a bisection.

- **MIN-CUT $\in$ NP-Complete**

$$3 - SAT \leq_P MAX - CUT \leq_P MAX - BIS \leq_P MIN - BIS$$

We show that maximum cut can be reduced to minimum bisection, thereby showing that minimum bisection is NP-complete. First note that maximum bisection can easily be reduced to minimum bisection (or vice-versa) because the maximum bisection of a graph is the minimum bisection of its complement. Note that this duality only works because the number of possible edges across a bisection is constant for a given graph; it fails for max cut and min cut, which is why it makes sense for one to be NP-complete and the other not. Now we show how to reduce max cut to max bisection. Given a graph G with n vertices, we claim that the width of the maximum cut for G is equal to that of the maximum bisection of the graph G' given by appending n isolated vertices to G. This is because any cut of G can be made into a bisection of G' by adding a suitable number of isolated vertices to each side of the cut. This gives an easy reduction of the decision problem for max-cut to that for max-bisection - the reduction is polynomial time since we only doubled the size of the input.

5.4 Algorithms for Graph Bisection

Many approximate algorithms have been proposed for solving the graph bisection problem. They can be broadly classified as

- Greedy Search Algorithms
- Spectral Methods
- Randomized Algorithms

5.4.1 Greedy Search Algorithms

Simple Greedy Method

The obvious greedy algorithm for the graph bisection problem consists of starting with any bisection (A, B) of V and computing a new bisection by swapping the pair of elements $a \in A, b \in B$, which maximizes the gain (number of edges cut before the swap minus the number after the swap). This process is repeated until the maximum gain is less than zero, or until the maximum gain is zero and another heuristic has determined that it is time to stop swapping "zero gain" pairs. The issue of breaking ties is resolved by choosing the pair to swap uniformly at random from the set of pairs for which the gain is maximum. To efficiently determine the cost of a swap, we store for each vertex $a \in A$ its internal cost $I(a)$, which is the number of edges $(a, a') \in E$ with $a' \in A$, and its external cost $E(a)$, which is the number of edges $(a, b) \in E$ with $b \in B$. This can be done in $O(m + n)$ time, and can be updated after swapping a and b in $O(deg(a) + deg(b))$ time. The gain in swapping a and b is then $gain = E(a) - I(a) + E(b) - I(b) - 2w(a, b)$, where $w(a, b) = 1$ if $(a, b) \in E$ and $w(a, b) = 0$ otherwise. We can therefore determine the best pair to swap in $O(n^2)$ time by simply running through all $(n/2)^2$ candidate pairs, keeping track of the leaders for the random choice at the end.

Kernighan-Lin Algorithm

Unfortunately, the greedy algorithm performs very poorly in practice, as it is common for graphs to have bisections which require several vertices to be swapped simultaneously before any gain is achieved, whereas the greedy algorithm can only search out bisections which are one swap away from the

current bisection. The Kernighan-Lin algorithm (KL) attempts to remedy this situation by using a greedy heuristic to search through bisections which are several swaps away from the current bisection. One step of the Kernighan-Lin algorithm consists of swapping k pairs of vertices ($1 \leq k \leq \frac{n}{2}$) which are chosen as follows:

- Make a copy of the graph.
- On the copy graph, swap the pair with the largest gain, even if this gain is negative, and mark the vertices as "swapped". Break ties randomly.
- Repeat the previous step on unmarked vertices until no points are left to be swapped.
- Pick k such that the cost of the bisection at the k^{th} step of the above process was smallest. Break ties (again) randomly.
- Swap these first k pairs of vertices on the original graph.

Since the inner loop at the k^{th} step of KL amounts to an update of the internal and external costs of the unfrozen vertices and a step of the greedy algorithm on a graph of size $(n - 2k)$, the total cost of running a step of KL is $T(2) + T(4) + ... + T(n) = O(n^3)$, where $T(l) = O(l^2)$ is the time required for one pass of the inner loop when l vertices are unmarked.

While the KL algorithm tends to produce fairly good results, it is fairly expensive, since each step can take up to $O(n^3)$. Fiduccia and Mattheyses[33] present an improvement to the basic KL algorithm, where each step (equivalent to KL step) takes $O(\mid E \mid)$. The Kernighan-Lin algorithm is shown in algorithm 1.

5.4.2 Spectral Methods

Simple Spectral Bisection

The basic spectral bisection method (due to Fiedler[34]) is a continuous relaxation method which recasts the problem of bisecting a graph as a problem of finding an eigenvector. We begin by observing that the transpose of the incidence matrix $I(G)^T$ has the property that for any vector $x \in \{-1, 1\}^n$, $I(G)^T x$

Algorithm 1 Kernighan-Lin Algorithm

Algorithm: Kernighan-Lin(G)
Input: $G = (V, E), \mid V \mid = 2n$
Output: Balanced bi-partition A and B with "small" cut cost.
begin
Bipartition G into A and B such that $\mid V_A \mid = \mid V_B \mid, V_A \cap V_B = \phi$,
and $V_A \cup V_B = V$
repeat
  Compute $D_v, \forall v \in V$.
  for $i = 1$ to n **do**
    Find a pair of unlocked vertices $v_{ai} \in V_A$ and $v_{bi} \in V_B$ whose
    exchange makes the largest decrease or smallest increase in cut cost;
    Mark v_{ai} and v_{bi} as locked, store the gain $\hat{g}_i$, and compute the new D_v,
    for all unlocked $v \in V$;
  end for
  Find k, such that $G_k = \sum_{i=1}^{k} \hat{g}_i$ is maximized;
  if $G_k > 0$ **then**
    Move $v_{a1}, \ldots, v_{ak}$ from V_A to V_B and $v_{b1}, \ldots, v_{bk}$ from V_B to V_A;
  end if
  Unlock v, $\forall v \in V$.
until $G_k \leq 0$;
end

is a vector in $\{-2, 0, 2\}^m$, a 0 appearing for each component corresponding to an edge in the graph for which both vertices have the same sign. (Recall that the incidence matrix $I(G)$ is an $n \times m$ matrix with column e containing a 1 in row i and a -1 in row j, where edge $c = (i, j) \in V$; in an undirected graph, the choice of which vertex gets which sign is arbitrary). As a result, $I(G)^T x^2 = x^T I(G) I(G)^T x = x^T L(G) x$ equals four times the number of edges $e = (i, j)$ such that $x_i x_j = -1$. The matrix $L(G) = I(G)I(G)^T$ is known as the Laplacian matrix, and can be characterized by

$$l_{ij} = \begin{cases} deg(i) & i = j \\ -1 & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

We observe that $L(G)$ is a symmetric matrix, so it has an orthonormal set of eigenvectors. Furthermore, we can see immediately that for any matrix G, $L(G)y = 0$ with $y = (1, \ldots, 1)^T$, hence y is an eigenvector of $L(G)$. As a result, $y^{\perp}$ is invariant under $L(G)$, and the spectrum $\sigma(L(G)) = \sigma(L(G)|_{y^{\perp}}) \cup \{0\}$.

Since $L(G)$ is positive semi-definite (recall $L(G) = I(G)I(G)^T$), we learn that the smallest eigenvalue of $L(G)|_{y^\perp}$ equals the second smallest eigenvalue of $L(G)$. Now we observe that $y^\perp$ contains the vector corresponding to any bisection, since it has an equal number of 1's and -1's; thus, the problem of finding the minimum bisection is exactly equivalent to the problem of finding the vector $x \in y^\perp \cap \{-1, 1\}^n$ which minimizes $x^T L(G)x$.

At this point we relax the requirement that $x \in \{-1, 1\}^n$ and find instead the vector $x \in y^\perp \cap \{x : \| x \| = \sqrt{n}\}$ which minimizes $x^T L(G)x$. But such an x has to be an eigenvector of $L(G)|_{y^\perp}$ corresponding to its smallest eigenvalue, which, as we mentioned above, is an eigenvector of $L(G)$ corresponding to its second smallest eigenvalue. Such an eigenvector can be found in polynomial time using a standard package, such as Matlab. Since this eigenvector will be in $y^\perp$, the sum of its components will be zero, so it has equal mass distributed above and below zero. The idea is to then round these components to ± 1 so that x is again in $\{-1, 1\}^n$. But of course, equal mass does not mean an equal number, so we change the rounding threshold from zero to the median value of the components of x, breaking ties randomly, in order to obtain a bisection. This approach produces both a lower bound and an upper bound on the minimum bisection – the lower bound comes from our knowledge of the minimum of $x^T L(G)x$ over all of $y^\perp \cap \{\| x \| = \sqrt{n}\}$, which properly includes all bisections, and the upper bound comes from the rounded result, which of course, can't be better than the actual result.

Boppana Bisection

We can generalize on the basic spectral bisection method to try to obtain tighter upper and lower bounds. Following Boppana [35], for any $d \in R^n$ we define the matrix $B = D - A$, where $D = diag(d)$ and A is the adjacency matrix ($a_{ij} = 1$ if $(i, j) \in E$, zero otherwise). Note that when $d_i = deg(i), B = L(G)$. We define the function

$$\begin{aligned} f(d, x) &= 2 \sum_{(i,j) \in E} (1 - x_i x_j) + \sum_{i \in V} d_i (x_i^2 - 1) \\ &= x^T (D - A)x - (\sum_{i \in V} d_i - 2m) \end{aligned}$$

and observe that if $x \in \{-1,1\}^n$, $\sum d_i(x_i^2 - 1) = 0$, and hence $f(d,x)$ equals four times the number of edges $e = (i,j)$ such that $x_i x_j = -1$. By the second formula for f, for any d the min-bisection problem amounts to finding the vector $x \in y^\perp \cap \{-1,1\}^n$ for which $x^T Bx$ is smallest. Unfortunately, $y^\perp$ is not invariant under B unless $d_i = deg(i)$, nor is B necessarily positive definite; it is, however, still easy to find the smallest eigenvector of $B|_{y^\perp}$ by considering the matrix PB, where P is the orthogonal projection onto $y^\perp$ Thus we may apply the same relaxation and rounding technique as before to obtain an upper and lower bound.

For each d, the lower bound is denoted by $g(d) = min_{x \in y^\perp, \|x\| = \sqrt{n}} f(d,x)$, and observe that g is bounded above by the cost of the min-bisection. It also turns out that g is concave and achieves its maximum for some (necessarily unique) finite d_{min}, so one can use the ellipsoid method to find this maximum in polynomial time. This gives the tightest lower bound available, and the hope is that the vector $x \in \{-1,1\}^n$ corresponding to the minimum bisection will be closer than any other bisection vector to the vector $x_{min} \in y^\perp \cap \{\| x \| = \sqrt{n}\}$ which minimizes $f(d_{min}, x)$, so that after rounding the solution can be recovered. This last idea (that the best rounded result will come from the best lower bound result) is given no justification in Bopanna.

5.4.3 Randomized Algorithms

Metropolis and Simulated Annealing

The Metropolis method has been successfully applied to the graph bisection problem. Jerrum and Sorkin[36] have shown that this method is very successful on $G_{n,p,r}$[1] graphs under suitable conditions on the parameters. In fact they prove that when $p - r = \Theta(n^{\delta-2})$ for $\delta > 11/6$, the Metropolis algorithm with high probability yields the minimal bisection in polynomial time. Their analysis is conducted using the acceptance function

$$\lambda(\delta cost) = \frac{exp(-\delta cost/T)}{1 + exp(-\delta cost/T)}$$

[1] $G_{n,p,r}$ is a probability distribution on graphs with vertex set $\{1,2,\ldots,n\}$ in which the presence of each possible edge is independent, with probability p for edges within $\{1,2,\ldots,n/2\}$ or $\{n/2+1,\ldots,n\}$ and probability $r < p$ for other edges

Genetic Algorithm

Another way to employ a stochastic search is through application of genetic algorithms. These algorithms search the solution space through simulating evolution, i.e. the "survival of the fittest strategy" They maintain a population of possible solutions, and generate new solutions through crossover and mutation. The "fittest" individuals have a greater chance of reproducing and surviving to the next generation, but the inferior solutions can also survive.

Within the GA framework, the solution to the graph bisection is represented as a vector of real values between 0 and 1, where each entry in the vector corresponds to a vertex in the graph. Partitioning of the graph corresponds to thresholding the vector at its median: the values above the median belong to the first partition, the values below the median belong to the second.

The selection of solutions to produce the future generations plays a very important role in a genetic algorithm. There are several schemes for selection: roulette wheel selection, scaling techniques, ranking methods, and tournaments.

The crossover function takes two parents, p_1 and p_2, and generates two children, c_1 and c_2. It is implemented as follows: select $V/2$ indices into the solution vector. In c_1 draw the values at these indices from the corresponding entries in p_1; the rest of the values comes from p_2. In c_2 the converse is the case: the values at the selected indices come from p_2 and the rest are taken from p_1. Note that as the population becomes more uniform, the crossover stops affecting the evolution: in a stable state, chances are that regardless whether the element comes from p_1 or p_2, it will fall in the same place with respect to the median.

To implement the mutation step, simply run the Metropolis algorithm for some number of steps. Also note that it is possible for the crossover operation to output a vector with repeated entries; if these occur at the median, the thresholding scheme is no longer guaranteed to generate a bisection. If this occurs, the mutation procedure will perturb the repeated entries, so that we can easily recover a full bisection.

It is interesting to note that running Simulated Annealing with multiple runners is very similar to running GA with an identity crossover, i.e. a crossover where c_1 is a copy of p_1 and c_2 is a copy of p_2.

5.5 Implementation Details

We selected the Kernighan-Lin Algorithm for our feature partitioning due to its simplicity and reasonable performance. Since the idea was to apply a graph bisection algorithm we used the METIS[2] and Chaco[3] packages that provide the graph partitioning implementation. It should be fairly easy to apply any other bisection algorithm.

[2]Software can be downloaded at http://www-users.cs.umn.edu/ karypis/metis/index.html
[3]http://www.cs.sandia.gov/ bahendr/chaco.html

Chapter 6
Related Work

This chapter discusses bits and pieces of other work related to text classification, using unlabeled data for classification.

6.1 Text Categorization

Text classification has been around in different forms for some time. One of the early application of text classification was to author identification. The seminal work by Mosteller and Wallace[37] examined authorship of the different Federalist papers using Bayesian analysis of features such as word and sentence length, frequency of function words, and vocabulary diversity. More recently text classification has been applied to a wide variety of practical applications: cataloging news articles [38] [39]; classifying web pages into a symbolic ontology [40]; finding a person's home page [12]; automatically learning the reading interests of users [41] [42]; automatically threading and filtering email by content [14]; and book recommendation [43]. An early and popular machine learning technique for text classification is naive Bayes. Its straightforward probabilistic nature has made it amenable to a variety of extensions. Limited word dependencies can be modeled using TAN trees [44]. Leverage of a class hierarchy can be provided through statistical shrinkage [45] or other more ad-hoc techniques [46]. The one-to-one class-to-component correspondence can be relaxed [47]. There are two different generative models that have been used for naive Bayes. The one used in this thesis is a multinomial (or in language modeling terms, "unigram") model, where the classifier is a mixture of multinomials and tracks the number of times a word appears in a document [17]. This formulation has been used by numerous practitioners of naive Bayes text classification [48] [47] [39]. A second formulation of naive Bayes text classification instead uses a generative model where each word in the vocabulary is a binary feature, and is modeled by a mixture of multi-variate Bernoullis [49] [50] [51] [46]. Empirical comparisons show that the multinomial formulation yields classifiers with consistently higher accuracy [17]. A variety of machine learning techniques other than naive Bayes have been applied to text classification. Support vector machines have recently

shown much promise [39]. Other approaches have used maximum entropy [52], neural nets [12] and several rule learning algorithms. Still others have used memory-based methods like k-nearest neighbor [53], and a variety of boosting approaches. To date, no single technique has emerged as clearly better than the others, though some recent evidence suggests that kNN and SVMs perform at least as well as other algorithms when there is a lot of labeled data for each class of interest [54]. Most studies into text classification use the simple document representation of bags-of-words, tracking the number of times each word occurs in a document, or even just whether or not it occurred. Consistently, efforts to include more substantial linguistic or semantic information have provided at most modest improvements to classification accuracy. Furnkranz et al. [55] uses shallow syntactic phrase patterns and finds some improvements to naive Bayes and rule learning algorithms. Mladenic selects variable-length phrases for text classification of web pages into the Yahoo hierarchy.

6.2 Expectation Maximization

The term Expectation Maximization was coined by Dempster et al in their seminal work [56]. The Expectation-Maximization (EM) algorithm is a general algorithm for maximum-likelihood estimation where the data are "incomplete" or unknown.

Definition 6.2.1 *We have a density function $p(\mathbf{x}|\Theta)$ that is governed by the set of parameters Θ (e.g., p might be a set of Gaussians and Θ could be the means and covariances). We also have a data set of size N, supposedly drawn from this distribution, i.e., $\mathcal{X} = \{\mathbf{x_1}, \ldots \mathbf{x_N}\}$. That is, we assume that these data vectors are independent and identically distributed (i.i.d.) with distribution p. Therefore, the resulting density for the samples is*

$$P(\mathcal{X}|\Theta) = \prod_{i=1} Np(\mathbf{x_i}, \Theta) = \mathcal{L}(\Theta|\mathcal{X})$$

This function $\mathcal{L}(\Theta|\mathcal{X})$ is called the likelihood of the parameters given the data, or just the likelihood function. The likelihood is thought of as a function of the parameters Θ where the data $\mathcal{X}$ is fixed. In the ***maximum likelihood estimation problem****, our goal is to find the Θ that maximizes $\mathcal{L}$. That is, we*

wish to find Θ^* *where*

$$\Theta^* = argmax\Theta\mathcal{L}(\Theta|\mathcal{X}).$$

Informally, the EM algorithm starts with randomly assigning values to all the parameters to be estimated. It then iteratively alternates between two steps, called the expectation step (i.e., the "E-step") and the maximization step (i.e., the "M-step"), respectively. In the E-step, it computes the expected likelihood for the complete data (observed data X and the hidden data Y) where the expectation is taken with respect to the computed conditional distribution of the hidden variables given the current settings of parameters and our observed (incomplete) data. Let $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$ be the complete data and assume a joint density function between the missing (hidden) and observed values. With this new density function we define a likelihood function $\mathcal{L}(\times|\mathcal{Z}) = \mathcal{L}(\times|\mathcal{X}, \mathcal{Y}) = \sqrt{(\S, \dagger|\times)}$ which is known as the likelihood of the complete data. The EM algorithm first finds the expected value of the complete data log-likelihood from the observed data and the current parameters estimates. The expected value can be formulated in the form of the following likelihood function

$$Q(\Theta, \Theta^{(i-1)}) = E[logp(\mathcal{X}, \mathcal{Y}|\Theta)|\mathcal{X}, \Theta^{(i-1)}]$$

where $\Theta^{(i-1)}$ are the current parameter estimates that are used for evaluating the above function and Θ is the parameter that we need to optimize to maximize the expected value.

In the M-step, the algorithm re-estimates all the parameters such that the likelihood function is maximized.

$$\Theta^{(i)} = argmax\mathcal{Q}(\Theta, \Theta^{(i-1)})\Theta$$

These two steps are repeated as necessary. Each iteration is guaranteed to increase the log-likelihood and the algorithm is guaranteed to converge to a local maximum of the likelihood function. Intuitively, what EM does is to iteratively "augment" the data by "guessing" the values of the hidden variables and to re-estimate the parameters by assuming that the guessed values are the true values. The EM algorithm is a hill-climbing approach, thus it can only be guaranteed to reach a local maxima. When there are multiple maximas,

whether we will actually reach the global maxima clearly depends on where we start; if we start with the right set of initial parameters, we will be able to find a global maxima. When there are multiple local maximas, it is often hard to identify the right set of initial parameters. "Expectation-maximization" is a description of a class of algorithms such as the one above, not a particular algorithm; EM is a recipe or meta-algorithm which is used to devise particular algorithms. The most common example of the the EM algorithm is for fitting a mixture density model. An EM algorithm can also find maximum a posteriori (MAP) estimates, by performing MAP estimation in the M step, rather than maximum likelihood.

6.2.1 Using EM for Text Classification

In order to use EM for text classification, the initial parameter estimation is performed by using the labeled data. The unlabeled data is used to iteratively adjust the parameters of the model.

Chapter 7
Experiments and Results

7.1 Introduction

This chapter outlines the experiments conducted in order to validate our hypothesis. Primarily, the tests conducted aimed at verifying the following two claims made in the previous chapters:

1. It is possible to construct partitions for datasets that do not possess a natural partitioning. Applying co-training using these constructed partitions should give better or at least the same performance as using random feature partitioning.

2. As a supplement to the previous claim, we wonder if natural splits are the best splits? Maybe, a constructed partition could give better performance in cases where the feature spaces are imbalanced.

3. Would the accuracy be affected if the feature space was partitioned into n partitions and these partitions are used with n classifiers?

The experiments were conducted on numerous text datasets that are commonly used. Results from our experiments validated some of our claims while the remaining claims were not conclusively backed by our experiments.

7.2 Datasets

Five dataset (and their subsets) where used for conducting experiments. These datasets were chosen due to their popular use as test datasets in various papers related to data mining or information retrieval. Even though co-training can be applied to a range of datasets (image, multimedia, etc.), our experiments have been restricted to text datasets due to their widespread availability and ease in extraction of features.

7.2.1 SpamAssassin Dataset

The SpamAssassin dataset[1] is a selection of messages which has been used for testing spam filtering systems. The dataset contains 3 parts - a set of spam

[1] http://spamassassin.org/publiccorpus/

messages, set of easy ham (non-spam) messages and a set of hard ham messages. The set of hard ham messages is difficult to classify due to similarity with the spam messages. The corpus contains around 1900 spam messages, 3900 ham messages and 250 hard ham messages. For our experiments, we used messages from the easy ham and spam sets.

7.2.2 20 Newsgroups Dataset

The 20 Newsgroups dataset[2] is a collection of approximately 20,000 Usenet newsgroup articles originally collected by Ken Lang. The dataset is partitioned almost evenly across 20 different newsgroups. The articles are typical postings and thus have headers including subject lines, signature files, and quoted portions of other articles. Some of the newsgroups are very closely related to each other (e.g. comp.sys.ibm.pc.hardware / comp.sys.mac.hardware), while others are highly unrelated (e.g misc.forsale / soc.religion.christian). The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering. The work by Nigam and Ghani[22], that laid the foundation of our work, also uses this dataset. To compare the experiments we think it is imperative to include this dataset in our experiments. Further details regarding the use of the dataset are along with the experiments conducted.

Table 2: List of the 20 newsgroups

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

7.2.3 WebKB Dataset

This dataset contains WWW-pages collected from computer science departments of various universities in January 1997 by the World Wide Knowledge Base (Web-¿KB)[3] project of the CMU text learning group. The 8,282 pages were manually classified into 7 categories student, faculty, staff, department,

[2]http://people.csail.mit.edu/u/j/jrennie/public_html/20Newsgroups/
[3]http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/

course, project, other. The class other is a collection of pages that were not deemed the "main page" representing an instance of the previous six classes. This dataset was used by Blum and Mitchell[9] in their work that introduced co-training.

7.2.4 7Sectors Dataset

The 7Sectors dataset[4] consists of 3417 html articles related to 7 industry sectors. The articles in each sector are further partitioned into sub-categories. The table below shows the distribution of articles across the different industry sectors.

Table 3: Characteristics of 7sectors dataset

Sector	Size	Proportion (in percentage)
basic	714	20.9
energy	265	7.7
financial	697	20.4
health	310	9.1
technology	823	24.1
transport	383	11.2
utilities	225	6.6

7.2.5 Oshumed Dataset

The Ohsumed corpus[5] was compiled by William Hersh. The corpus contains around 50,000 documents which are medical abstracts and have a high percentage of medical terms. These documents have been assigned to one or multiple of the 23 MeSH (Medical Subject Headings) "diseases" categories. The abstracts in the documents are very difficult to comprehend for people from non-medical background. This is a distinguishing feature of this corpus. Our experiments were conducted on a subset of this corpus.

7.3 Experimental Setup

Our text categorization system (see appendix) was used to perform the experiments. The system has modules for feature extraction, data cleaning and pre-processing. The keywords in the documents were extracted using either a

[4]http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/bootstrappingIE/7sectors.tar.gz
[5]ftp://medir.ohsu.edu/pub/OHSUMED/

MIME parser or an HTML parser. For the SpamAssassin and 20 Newsgroups datasets all the headers in the messages were removed. For HTML documents (WebKB and 7Sectors dataset), the meta tags, href links and any presentation related tags (font, size, color, etc.) were removed. Stemming was not used since our initial experiments did not show any advantage with stemming enabled. Keywords with very low frequencies were eliminated. Keywords with smaller lengths were removed to eliminate keywords like 'a', 'the', 'an', 'if', etc. To make documents of different lengths comparable, feature vectors are normalized to unit length. A Naïve Bayes classifier was used as the classification model. Error rate was used as the evaluation metrics.

$$\begin{aligned} Errorrate &= 1 - Accuracy \\ Accuracy &= \frac{TP + TN}{TP + TN + FP + FN} \end{aligned} \tag{2}$$

where
TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative The good property of error rate is that its symmetric measure in the sense that it takes into account both the true positive and the true negative. Some equivalent measures consider only the true positive. To test our hypothesis regarding improved classification using a constructed feature split, the following test was setup:

1. Construct a dataset (D) by combining n disjoint/unrelated datasets $(d_1, d_2, \ldots d_n)$.

2. Partition the features in D into n sets $F_1, F_2, \ldots F_n$, using any feature splitting technique. We used graph partitioning.

3. Train n classifier $(C_1, C_2, \ldots C_n)$ on the input data such that classifier C_1 uses F_1 and so on. These n classifiers form the combined co-trained classifier (call it $C_{construct}$)with constructed feature sets.

4. From the n disjoint datasets, co-train n classifiers. In this case, the features used by classifiers $C_1, C_2, \ldots C_n$ will be strictly from datasets $d_1, d_2, \ldots, d_n$ respectively. Lets call this co-trained classifier C_{known} since the classifier is build on known partitions of the dataset.

5. Compare the error rates obtained $C_{construct}$ and C_{known}. If $C_{construct}$ performs as well as C_{known} then we can claim that co-training based on constructed feature partitions performs as good as co-training based on natural or known feature partitions.

Our results show that co-training based on constructed feature partitions performs as good as or better than natural or known partitions. To further emphasize the advantage of constructing feature splits, we performed experiments on datasets with natural partitions that were imbalanced. SpamAssassin is a good example of such a dataset, wherein, the features in the 'Subject' partition are much less than the features in the 'Body' partition. Our experiments show that for imbalanced datasets, co-training with constructed datasets performs particularly better than co-training on natural partitions. After the first set of experiments provided positive results, we designed another set of experiments to see how co-training based on constructed feature partitions works on datasets that have no known partitions. We compare our results against randomly created partitions. The following is a summary of the steps involved in the tests:

1. Partition the features in the dataset into n sets $F_1, F_2, \ldots F_n$, using graph partitioning.

2. Train n classifier $(C_1, C_2, \ldots C_n)$ on the input data such that classifier C_1 uses F_1 and so on. These n classifiers form the combined co-trained classifier (call it $C_{construct}$)with constructed feature sets.

3. Now partition the features in the dataset randomly into n partitions. Build a co-trained classifier C_{random} using these partitions.

4. Run the classifiers $C_{construct}$ and C_{random} on the test dataset and compare their error rates. If $C_{construct}$ performs better than C_{random} then we can claim that co-training based on constructed feature partitions performs well even when the partitions are not known.

Random partitioning is used as the benchmark because that is the least one can do when the partitions are unknown. The final set of experiments aimed at comparing our modification to co-training against the use of Expectation Maximization as a technique for classification in the presence of missing data. The following section describes each experiment in detail followed by the results obtained.

7.4 Results

In order to compare partitioning based co-training and co-training with known partitions, two experiments were performed. The first experiment was conducted on a subset of the 20 Newsgroups dataset. The dataset consists of 50 labeled documents and 250 unlabeled documents of each class. Each document in the dataset is constructed by combining messages from 3 feature sets (FS_1, FS_2 and FS_3). The dataset is described in table 4. Observe that, intuitively each feature set consists of keywords that belong to one topic or concept. For example, FS_1 consists of keywords from rec.sport.baseball and rec.sport.hockey. Since both newsgroups talk about sports, the keywords would be related. At the same time, since the two newsgroups talk about specific sports (hockey and baseball in this case), they will still differ in the terms used. Hence they can be considered to belong to separate classes. This reasoning can be applied to FS_2 and FS_3 also. The assignment of documents that contain

$$rec.sport.baseball + sci.med + talk.politics.guns$$

to class 1 is arbitrary. The idea was to have documents that are derived from the same feature sets but still different enough to belong to different classes. Let us call this dataset D1. Each labeled document is partitioned into 3 sets using the graph partition technique described in the previous chapter. In doing so, we ignore the fact that each document is constructed by combining messages from three newsgroups. Each partition obtained using the graph partitioning technique becomes the feature set for one classifier in the co-training combination. Unlabeled documents are classified by each of the 3 classifiers and the final class of the document is obtained by equally weighing the probabilities obtained from each individual classifier. During each iteration, a user defined number of messages from the unlabeled set are added to the labeled set. The error rates

are recorded for each iteration. Error rates are also obtained for co-trained classifier based on the known feature sets. Figure 4 compares the error rates for the two classifiers. The classifier with constructed feature splits has lower error rate as compared to one with random feature splits. Figure 4 shows the error

Table 4: Dataset D1 Subset of 20 Newsgroups

	FS_1	FS_2	FS_3	Labeled	Unlabeled	Test
Class 1	rec.sport.baseball	sci.med	talk.political.guns	50	250	250
Class 2	rec.sport.hockey	sci.space	talk.political.mideast	50	250	250

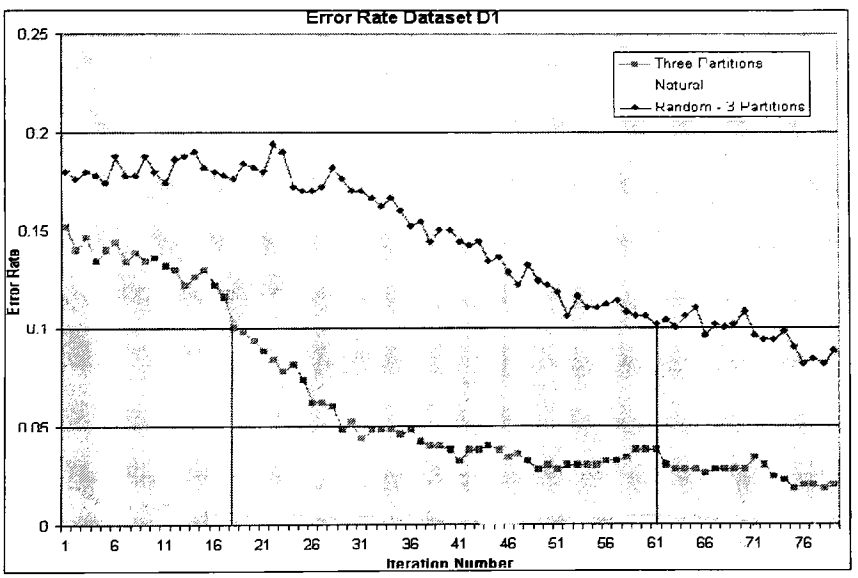


Figure 4: Error rate versus iterations for Dataset 1

rates for the co-training based on natural, random and constructed partitions. As expected natural partitioning outperforms the other two. But constructed partitioning outperforms random partitioning by a substantial margin. Two important observations can be made from these results:

- Constructed partitioning catches up with natural partitioning much sooner than random partitioning does.
- If a minimum error rate threshold is required by the classifier constructed partitioning reaches this threshold in much fewer iterations. Since each iteration corresponds to certain number of labeled documents, we can effectively say that constructed partitioning requires much less (approximately 3.5 times) labeled documents.

We ground our conclusions we performed the same experiment on another dataset 7Sectors. A labeled document is constructed similar to dataset 1. The feature partitions for the constructed dataset are shown in table 5. For

Table 5: Dataset D2 - Subset of 7Sectors

	FS_1	FS_2	FS_3	Labeled	Unlabeled	Test
Class 1	Communications equipment industry	Gold and silver industry	Air courier industry	10	40	40
Class 2	Office equipment industry	Iron and steel industry	Airline industry	10	40	40

this dataset, the error rates are shown in figure 5. It can be noticed that both natural partitioning and constructed partitioning start at the same error rates. For initial iterations, constructed partitioning performs better than natural partitioning. At iteration 10, natural partitioning is as good as constructed partitioning. This indicates that a few wrong guesses by the constructed classifier substantially affect its performance. Thus, we can conclude that the constructed partitioning is sensitive to errors. Beyond iteration 10, the natural partitioning rapidly outperforms constructed partitioning. At the same time, constructed partitioning still performs better than random partitioning. For both the above experiments, outputs are not shown for all iterations since the curves do not display any significant changes in the subsequent iterations. The objective of the above two experiments was to show that constructed partitioning outperforms random partitioning. We could emphatically state that since we had prior knowledge regarding the number of natural partitions. If this knowledge was not available then we could not conclusively make the above claim. If we did not know the number of natural partitions, we would have had to make a guess

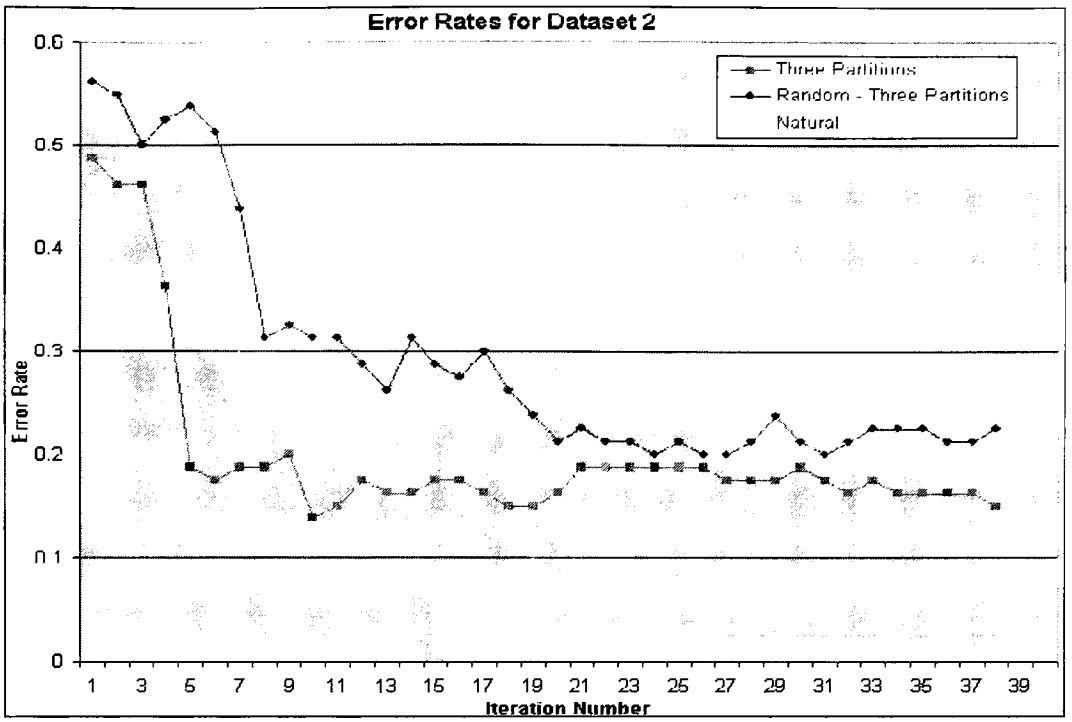


Figure 5: Error rate versus iterations for Dataset 2

on the number of natural partitions and then compare the error rates for natural and constructed partitioning for those number of partitions. The next step after concluding that constructed partitions outperform random partitions is to compare the two partitioning techniques on datasets for which natural partitions are not known. We again picked a subset of the 20 Newsgroups dataset. The dataset is shown in table 6. In this dataset there is a single feature set which intuitively relates to keywords in the sports category. This dataset was partitioned into N ($N = 2$ to 10) partitions using random and graph based partitioning. For each partitioning, N classifiers are co-trained. Figure 6 compares random and graph based partitioning for $N = 3$. For all values of N, graph based partitioning outperformed random partitioning. Similar test was performed on the Oshumed dataset. The constructed dataset is shown in table 7. Class one corresponds to abstracts that talk about heart diseases and whereas class two has abstracts related to diseases not related to heart. Both classes are

Table 6: Dataset D3 - Subset of 20 Newsgroups

	FS_1	Labeled	Unlabeled	Test
Class 1	rec.sport.baseball	50	250	250
Class 2	rec.sport.hockey	50	250	250

derived from the common feature set that corresponds to the general category of diseases. The results on this dataset are contrary to expectation. Most of

Table 7: Dataset D4 - Subset of Ohsumed dataset

	FS_1	Labeled	Unlabeled	Test
Class 1	Heart related abstracts	25	500	500
Class 2	Non-heart related abstracts	25	500	500

the times, for different values of N, random partitioning outperformed graph based partitioning. Results for $N = 3$ are shown in figure 7. On further investigating, we noticed certain peculiar characteristics of this dataset that lead to such an anomalous behavior. The dataset consists of numerous keywords having the same meaning. Each author has used a specialized term for the same concept resulting in higher number of unique terms. Terms such as 'bolus' and 'dosage' are synonyms of each other. Similarly, 'diazepam' and 'anxiolytic' are synonyms for anxiety relieving drugs. This quality of the dataset throws of the graph partitioning technique, since it relies on the degree of similarity between the keywords. The use of synonyms hides this similarity and the partitions generated are weaker. Apart from the Ohsumed dataset, all the other datasets displayed favorable results for graph based partitioning as compared to random partitioning. We wanted to verify if constructed partitioning is a better alternative for datasets which are imbalanced. Email messages are a good example of such a dataset, where the two feature sets - message body and message subject are imbalanced in the number of features. We compared co-training based on naturally available partitions against constructed partitions. This test was performed on the WebKB dataset. The characteristics of the dataset are summarized in table 8. In this dataset the web pages are divided into two classes course related web pages and non-course related web pages. The two feature

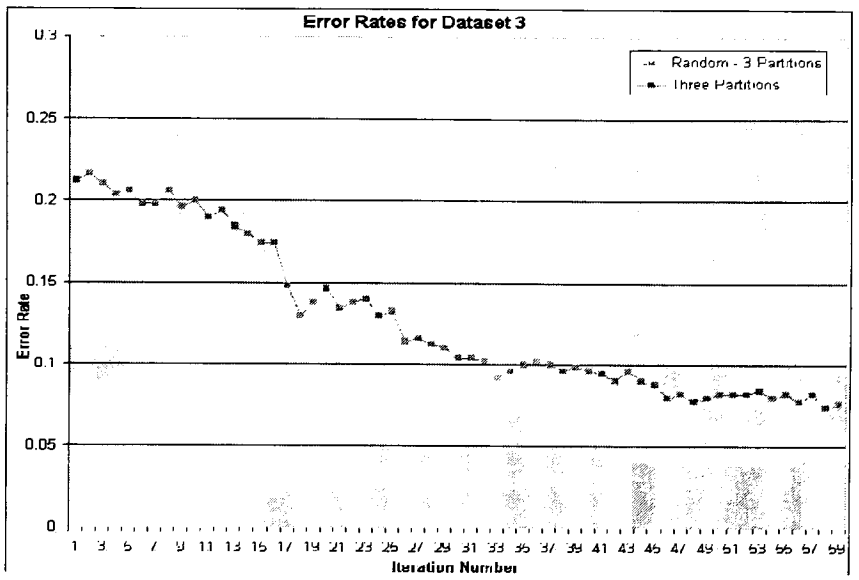


Figure 6: Error rate versus iterations for Dataset 3

sets correspond to keywords in the message body and keywords in the message subject. The results on this dataset are shown in figure 8. The results show

Table 8: Dataset D5 Subset of WebKB dataset

	FS_1	FS_2	Labeled	Unlabeled	Test
Course related	Text in pages	Text in links	25	100	100
Non-course related	Text in pages	Text in links	25	100	100

that graph based partitioning outperforms both random and natural partitioning. Notice that the initial error rate for random and graph based are less than natural partitioning. With every iteration, the error rate for graph based partition decreases rapidly. The plot for random partitioning has many points at which the error rate increases. This is the drawback of random partitioning.

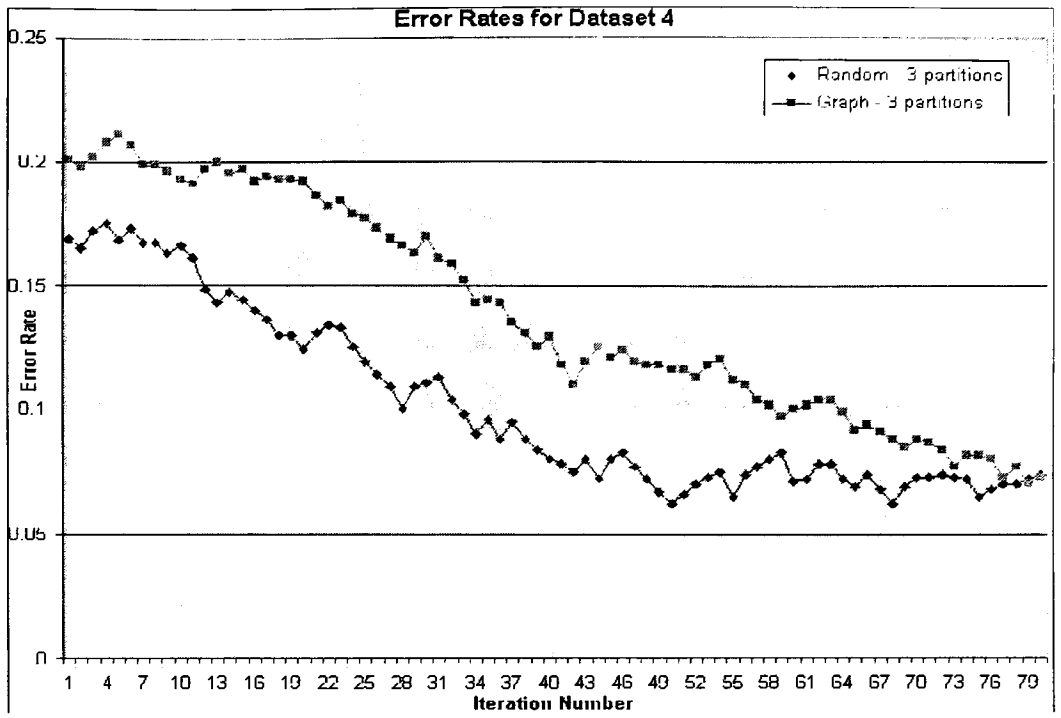


Figure 7: Results for dataset 4 for $N = 3$

We can also notice that the plots for random and graph based have a similar slope. This is indicative of equal sized partitions in both these techniques.

Some of our key observation during our experiments have been as follows:

- Constructed splits based co-training results starts with a lower error rate.
- Co-training based on constructed splits continues to decrease the error rate with each iteration as compared to random splits based co-training. We can say that the number of labeled documents required to reach a pre-specified threshold is less for co-training based on constructed splits. This amounts to better performance at a lower cost both in terms of number of labeled documents and the time required.

Finding PAC-style bounds on the number of labeled documents for co-training with partitioning is a new problem in itself. Coming up with such bounds remains the motivation for continuing this work beyond this thesis.

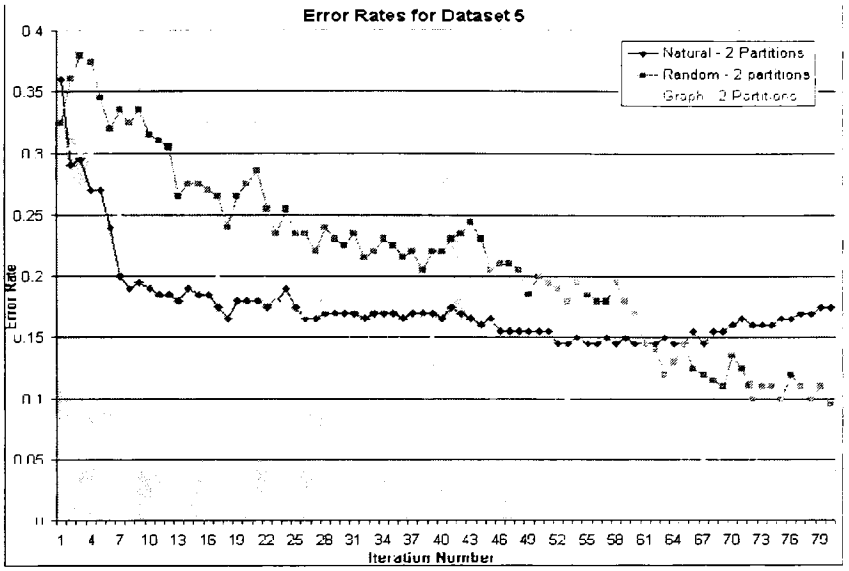


Figure 8: Comparison of error rates between natural, random and graph based partitioning with 2 partitions

7.4.1 K-training Results

After observing that feature partitioning leads to better performance, the natural question that occurred was whether increasing the number of splits can increase the accuracy of a co-trained classifier. We setup experiments to verify this hypothesis. The first dataset consisted of seven sets of newsgroups from the 20 newsgroups dataset. Each set consisted of two newsgroups that are similar in their vocabulary and content. A document in the dataset is a group of seven messages, one from each group. Labeled documents are divided into two classes. Unlabeled documents are constructed in a similar fashion. 25 labeled documents and 100 unlabeled documents were used. The features in each document are split into N partitions. For each partition size, equivalent number of classifiers are co-trained. The error rate for each value of N is computed. Similarly, the natural split of the features in the documents is obtained easily from the seven

groups mentioned earlier. Seven classifiers are co-trained using these natural feature splits. The error rates for values of N ranging from 2 to 7 is shown in figure 9. The error rate for natural partitioning based co-training is also shown. Notice that with increasing number of partitions the accuracy (= 1 error

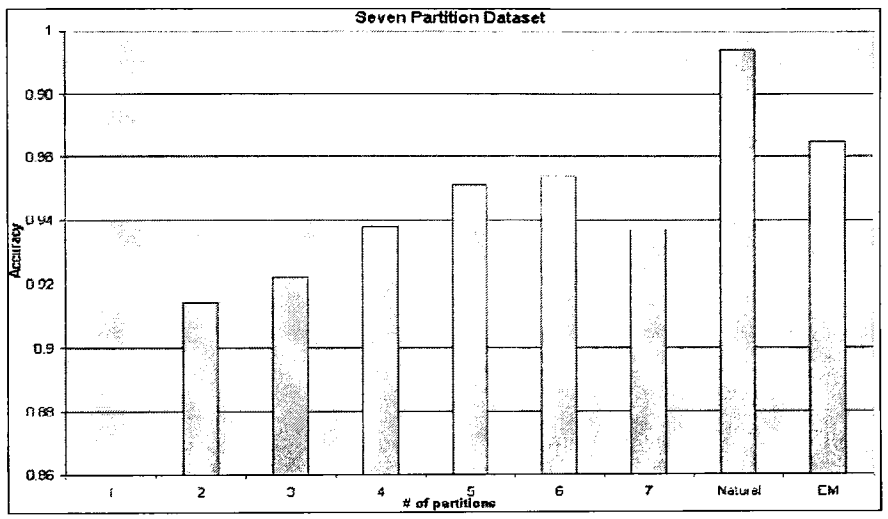


Figure 9: Number of partitions versus accuracy for the dataset with seven partitions

rate) increases, till the number of partitions reaches 6. Beyond six partitions the accuracy starts to drop. The best accuracy of 95.5% was observed for six partitions, while the accuracy of co-training based on natural partitions is 99.6%. Let us outline the results on the second dataset before we can draw any conclusions. The second dataset was build similar to the one above, but instead of seven newsgroups we used just 3 newsgroups. The results are shown in figure 10. The peak accuracy was observed with three partitions, which is the number of natural partitions in the dataset. We conducted many more experiments but the above two cases are representative of the general tread

observed. Hence we are omitting the remaining results. It can be observed that as the number of natural partitions increases, the results obtained by feature partitioning became unpredictable and in accurate. This can be observed in the first dataset where the peak was observed at six partitions. For smaller number of natural partitions, the accuracy generally peaks when the number of partitions is equal to the number of natural partitions. This behavior indicates that the dataset is able to exploit the existence of these features splits. This behavior can be observed in the results of the second experiment. Under no circumstances are we able to come up to the accuracy obtained from natural splits. On the whole, we cannot conclusively state that increasing the number of partitions results in an improved classifier. On the other hand we somewhat observe the number of partitions versus accuracy profile as hypothesized in figure 2 on page 27. The reason for poor accuracy with large number of partitions can be attributed to the weakness of the graph partitioning technique with increasing number of partitions. The graph partitioning technique becomes unstable with higher number of partitions.

Comparison with Expectation Maximization

After comparing our technique with random partitioning (which is a relatively naive technique), the stage is set to compare our technique to a more widely acclaimed technique. Recently, the Expectation Maximization (EM) technique has been used successfully for building classifiers in the presence of unlabeled data. The EM algorithm has been described in section 6.2. The algorithm is iterative and consists of two steps the E (Expectation) step and the M (Maximization) step. EM experiments are conducted on the same dataset as the k-training experiments. Results are shown in figures 9 and 10. For most of the datasets, we observed that our best time (corresponding to the optimal number of partitions) was better than the best time for EM.

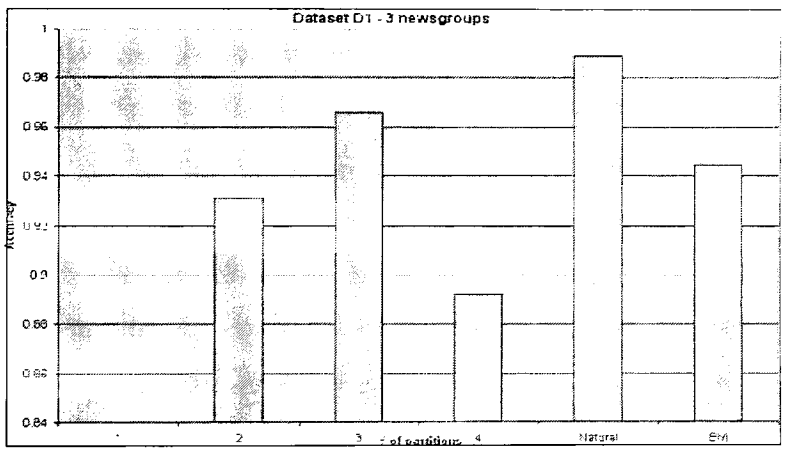


Figure 10: Number of partitions versus accuracy for the dataset with three partitions

Chapter 8
Conclusion

This primary objective of the thesis was to analyze and extend a technique that handles classification in the presence of unlabeled data. Co-training exploits the separation between the features, along with the information hidden in unlabeled data, to build a classifier that has accuracy even higher than a classifier with all labeled data. Possibility of further improving this accuracy using intelligent feature partitioning techniques, was the motivation behind the work in this thesis. We have experimentally shown that constructing feature partitions for co-training results in better accuracy as compared to natural or random feature splits. We go a step further by proposing the k-training paradigm in which the features are partitioned into k partitions and a co-trained classifier is built on these partitions. Though we were conclusively able to say that k-training has a benefit over co-training, there is definitely scope for further exploration. The following section touches upon some of the areas we think could be improved or added to the current state of the thesis.

8.1 Future Work

There is a trade-off between time and accuracy while building a co-trained classifier. A co-trained classifier takes much more time than a standard classifier. This difference is understandable because the co-trained classifier has to update the model at each iteration by classifying the unlabeled data. We would like to look at techniques that could make the co-trained classifier faster to build. This may be achieved by incorporating incremental learning techniques.

We were not conclusively prove that k-training would be beneficial for all datasets or under all scenarios. We need to develop a theoretical foundation that validates our experimental results. Still a little skeptical about k-training. At the same time, the vast number of applications of this technique motivates us to explore further. K-training has huge promise in sensor networks, image data and any other sources of data that have multiple modalities.

Some of the immediate work would explore the possibility of using other classifiers, for example, an SVM classifier. We would also like to cover a wider

range of datasets for our experiments. For the k-training setting, we suspect that the graph partitioning algorithm adversely affected our results for partitions greater than two. We are keen on exploring other graph partitioning techniques or even other feature partitioning techniques. Another drawback of the k-training approach is that the value of k has to be chosen arbitrary or we have to try a range of values for k. We want to incorporate techniques that would give us an estimate of the value of k. We also intend to use measures of correlation between features other than mutual information.

We think that this thesis addresses very interesting issues that have real applications and we are excited to carry this work further.

List of References

[1] T. Zhang and F Oles, "A probability analysis on the value of unlabeled data for classification problems," 2000.

[2] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabeled documents using em," *Machine Learning*, vol. 39, no. 2–3, pp. 103–134, 2000.

[3] B. Liu, Y. Dai, X. Li, W. S. Lee, , and P. Yu., "Building text classifiers using positive and unlabeled examples," 2003.

[4] T. Joachims, *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms.* Kluwer Academic Publishers, 2002.

[5] K. Bennett and A. Demiriz, "Semi-supervised support vector machines," pp. 368–374, MIT Press, 1998.

[6] H. Seung, M. Opper, , and H. Sompolinsky, "Query by committee," pp. 287–294, ACM Press, 1992.

[7] A. McCallum and K. Nigam, "Employing em and pool-based active learning for text classification," pp. 259–267, 1998.

[8] K. Bennett, A. Demiriz, and R. Maclin, "Exploiting unlabeled data in ensemble methods," pp. 289–296, ACM Press, 2002.

[9] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," pp. 92–100, COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1998.

[10] F. G. Cozman and I. Cohen, "Unlabeled data can degrade classification performance of generative classifiers," pp. 327–331, 2002.

[11] M. Craven, D. DiPasquo, , D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to construct knowledge bases from the world wide web," vol. 118, pp. 69–113, Artificial Intelligence, 2000.

[12] J. Shavlik and T. Eliassi-Rad, "Intelligent agents for web-based tasks: An advice-taking approach," pp. 63–70, Papers from the AAAI Workshop on Learning for Text Categorization, AAAI Press, 1998.

[13] M. J. Pazzani, J. Muramatsu, and D. Billsus, "Syskill & webert: Identifying interesting web sites," pp. 54–59, Proceedings of the Thirteenth National Conference on Artificial Intelligence, 1996.

[14] M. Sahami, S. Dumais, D. Heckerman, , and E. Horvitz, "On bayesian spam filtering: A bayesian approach to filtering junk email," AAAI Workshop on Learning for Text Categorization, 1998.

[15] F. Peng, D. Schuurmans, and S. Wang, "Augmenting naive bayes classifiers with statistical language models," *Information Retrieval*, vol. 7, no. 3-4, pp. 317–345, 2004.

[16] M. E. Maron, "Automatic indexing: An experimental inquiry," *Journal of ACM*, vol. 8, no. 3, pp. 404–417, 1961.

[17] A. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," pp. 41–48, In AAAI/ICML-98 Workshop on Learning for Text Categorization, 1998.

[18] T. Li, S. Zhu, Q. Li, and M. Ogihara, "Gene functional classification by semi-supervised learning from heterogeneous data," pp. 78–82, Proceedings of 18th ACM SAC Bioinformatics Track, 2003.

[19] M. Collins and Y. Singer, "Unsupervised models for named entity classification," In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, 1999.

[20] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," pp. 189–196, In Proceedings of the Annual Meeting of the Association for Computational Linguistics, 1995.

[21] S. Kiritchenko and S. Matwin, "Email classification with co-training," Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research, 2001.

[22] K. Nigam and R. Ghani, "Understanding the behavior of co-training," Proceedings of Workshop on Text Mining at Sixth SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000.

[23] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[24] T. M. Mitchell, *Machine Learning.* New York: McGraw-Hill, 1997.

[25] D. Haussler, "Quantifying inductive bias: Ai learning algorithms and valiant's learning framework," vol. 36, pp. 177–221, 1988.

[26] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic coclustering," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 89–98, ACM Press, 2003.

[27] G. Das, H. Mannila, and P. Ronkainen, "Similarity of attributes by external probes.," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.*, pp. 23–29, 1998.

[28] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 207–216, ACM Press, 1993.

[29] I. S. Dhillon, S. Mallela, and R. Kumar, "A divisive information theoretic feature clustering algorithm for text classification," *Journal of Machine Learning Research: Special Issue on Variable and Feature Selection*, vol. 3, pp. 1265–1287, 2003.

[30] J. Li and H. Zha, "Simultaneous classification and feature clustering using discriminant vector quantization with applications to microarray data analysis," in *IEEE Computer Society Bioinformatics Conference*, 2002.

[31] Y. Yang and C. G. Chute, "An example-based mapping method for text categorization and retrieval," *ACM Trans. Inf. Syst.*, vol. 12, no. 3, pp. 252–277, 1994.

[32] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*, pp. 47–63, ACM Press, 1974.

[33] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th conference on Design automation*, pp. 175–181, IEEE Press, 1982.

[34] M. Fiedler, "Algebraic connectivity of graphs," *Czech. Mathematical Journal*, vol. 23, pp. 298–305, 1973.

[35] R. Boppana, "Eigenvalues and graph bisection: An average case analysis," in *28th Annual Symposium on Foundations of Computer Science*, 1987.

[36] M. Jerrum and G. Sorkin, "Simmulated annealing for graph bisection," in *Annual Symposium on Foundations of Computer Science*, pp. 94–103, 1993.

[37] F. Mosteller and D. L. Wallace, *Inference and Disputed Authorship: The Federalist.* Reading, Massachusetts: Addison-Wesley, 1964.

[38] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, pp. 3–12, Springer Verlag, Heidelberg, DE, 1994.

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

[40] M. Craven, D. DiPasquo, D. Freitag, A. K. McCallum, T. M. Mitchell, K. Nigam, and S. Slattery, "Learning to construct knowledge bases from the world wide web," *Artificial Intelligence*, vol. 118, no. 1/2, pp. 69–113, 2000.

[41] M. J. Pazzani, J. Muramatsu, and D. Billsus, "Syskill webert: Identifying interesting web sites," in *AAAI/IAAI, Vol. 1*, pp. 54–61, 1996.

[42] K. Lang, "Newsweeder: learning to filter netnews," in *Proceedings of the 12th International Conference on Machine Learning*, pp. 331–339, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995.

[43] R. J. Mooney and L. Roy, "Content-based book recommending using learning for text categorization," in *Proceedings of DL-00, 5th ACM Conference on Digital Libraries*, pp. 195–204, ACM Press, New York, US, 2000.

[44] M. Sahami, "Learning limited dependence bayesian classifiers," in *Second International Conference on Knowledge Discovery in Databases*, 1996.

[45] A. K. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng, "Improving text classification by shrinkage in a hierarchy of classes," in *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pp. 359–367, Morgan Kaufmann Publishers, San Francisco, US, 1998.

[46] D. Koller and M. Sahami, "Hierarchically classifying documents using very few words," in *Proceedings of ICML-97, 14th International Conference on Machine Learning*, (Nashville, US), pp. 170–178, Morgan Kaufmann Publishers, San Francisco, US, 1997.

[47] H. Li and K. Yamanishi, "Document classification using a finite mixture model," in *Proceedings of the 35th conference on Association for Computational Linguistics*, pp. 39–47, Association for Computational Linguistics, 1997.

[48] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," pp. 3–12, 1994.

[49] S. Robertson and K. Sparck-Jones, "Relevance weighting of search terms.," *Journal of the American Society for Information Science*, vol. 27, no. 3, pp. 129–146, 1976.

[50] D. D. Lewis, "An evaluation of phrasal and clustered representations on a text categorization task," in *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 37–50, ACM Press, 1992.

[51] L. S. Larkey and W. B. Croft, "Combining classifiers in text categorization," in *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval* (H.-P. Frei, D. Harman, P Schauble, and R. Wilkinson, eds.), (Zurich, CH), pp. 289–297, ACM Press, New York, US, 1996.

[52] K. Nigam, J. Lafferty, and A. McCallum, "Using maximum entropy for text classification," 1999.

[53] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 412–420, Morgan Kaufmann Publishers Inc., 1997.

[54] Y. Yang, "An evaluation of statistical approaches to text categorization," *Inf. Retr.*, vol. 1, no. 1-2, pp. 69–90, 1999.

[55] J. Furnkranz, T. Mitchell, and E. Riloff, "A case study in using linguistic phrases for text categorization on the www," 1998. Working Notes of the AAAI/ICML Workshop on Learning for Text Categorization.

[56] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," vol. 1, pp. 1–38, 1977.