2007-12-21

Induction of Classifiers from Multi-labeled Examples: an Information-retrieval Point of View

Kanoksri Sarinnapakorn
University of Miami, ksarin@miami.edu

UNIVERSITY OF MIAMI

INDUCTION OF CLASSIFIERS FROM MULTI-LABELED EXAMPLES: AN INFORMATION-RETRIEVAL POINT OF VIEW

By

Kanoksri Sarinnapakorn

A DISSERTATION

Submitted to the Faculty
of the University of Miami
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Coral Gables, Florida

December 2007

UNIVERSITY OF MIAMI

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

INDUCTION OF CLASSIFIERS FROM MULTI-LABELED EXAMPLES: AN INFORMATION-RETRIEVAL POINT OF VIEW

Kanoksri Sarinnapakorn

Approved:

SARINNAPAKORN, KANOKSRI (Ph.D., Electrical and Computer Engineering)
Induction of Classifiers from Multi-labeled Examples: an Information-retrieval Point of View (December 2007)

Abstract of a dissertation at the University of Miami.

Dissertation supervised by Professor Miroslav Kubat.
No. of pages in text. (168)

An important task of information retrieval is to induce classifiers capable of categorizing text documents. The fact that the same document can simultaneously belong to two or more categories is referred by the term *multi-label classification* (or *categorization*). Domains of this kind have been encountered in diverse fields even outside information retrieval. This dissertation discusses one challenging aspect of text categorization: the documents (i.e., training examples) are characterized by an extremely large number of features. As a result, many existing machine learning techniques are in such domains prohibitively expensive. This dissertation seeks to reduce these costs significantly.

The proposed scheme consists of two steps. The first runs a so-called *baseline induction algorithm* (BIA) separately on different versions of the data, each time inducing a different subclassifier—more specifically, BIA is run always on the same training documents that are each time described by a different subset of the features. The second step then combines the subclassifiers by a *fusion algorithm*: when a document is to be classified, each subclassifier outputs a set of class labels accompanied by its confidence in these labels; these outputs are then combined into a single multi-label recommendation. The dissertation investigates a few alternative fusion techniques, including an original one, inspired by the Dempster-Shafer Theory. The main contribution is a mechanism for assigning the mass function to individual labels from subclassifiers.

The system's behavior is illustrated on two real-world data sets. As indicated, in each of them the examples are described by thousands of features, and each example is labeled with a subset of classes. Experimental evidence indicates that the method can scale up well and achieves impressive computational savings in exchange for only a modest loss in the classification performance. The fusion method proposed is also shown to be more accurate than other more traditional fusion mechanisms.

For a very large multi-label data set, the proposed mechanism not only speeds up the total induction time, but also facilitates the execution of the task on a small computer. The fact that subclassifiers can be constructed independently and more conveniently from small subsets of features provides an avenue for parallel processing that might offer further increase in computational efficiency.

To my beloved parents

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this dissertation. First, thanks in large part to the kindness and considerable mentoring provided by Prof. Miroslav Kubat, my dissertation advisor. His help, advice, and encouragement guided the way for this long journey. He has absolutely been a pleasure to work with. I want to offer my heartfelt thanks to committee members, Prof. Moiez A. Tapia, Prof. Akmal A. Younis, Prof. Nigel M. John, and Prof. Maria M. Llabre who gave insightful comments and invaluable suggestions to improve this research.

This work would not have been possible without the financial support of the Behavioral Medicine Research Center (BMRC) and the Department of Psychology. I am especially indebted to Prof. Barry E. Hurwitz, Prof. Philip M. McCabe, Prof. A. Rodney Wellens, and particularly Prof. Maria M. Llabre. Prof. Llabre's patience, and wisdom are truly remarkable. I thank her for sharing them so generously with me along the way.

Special appreciation is due Prof. Kamal Premaratne, for it was his class that introduced me to data fusion and led to an important part of work described in this dissertation. He has given me kind advices and suggestions throughout my Ph.D. study. Prof. Mei-Ling Shyu also deserves a great deal of thanks. Through her a new chapter in life was opened to me.

I am grateful to Dr. Jan Zizka from Masaryk University, Brno, Czech Republic, and to Dr. Ralf Steinberger and Dr. Bruno Pouliquen from the European Commission Joint Research Centre (JRC), Ispra site, Language Technology Group for providing the EUROVOC data.

I thank my colleagues at BMRC and friends at UM Thai Student Organization, the Department of Electrical and Computer Engineering, and the Department of Psychology (You know who you are!) for their support and encouragement. In particular, special thanks go to Omaima Nomir, my best friend and Dr. Feng Zhao, my former supervisor at BMRC.

Nobody has been more important to me in the pursuit of this project than the members of my family. I would like to thank my parents whose love and guidance are with me in whatever I pursue, my brothers and their families who have taken good care of parents during these years, and my sister who gives me continuing support and is always with me whenever I need help.

Contents

Notation

Θ	frame of discernment
$Acc(c)$	accuracy of a classifier in classifying class c
$Bel(.)$	belief function, $Bel(.) \in [0,1]$
C	total number of categories
$f(.,.)$	ranking function or confidence measure, $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$
$f^*(.,.)$	normalized confidence measure, $f^*(.,.) \in [0,1]$
$\mathfrak{F}(\Theta)$	set of focal elements of Θ, $\mathfrak{F}(\Theta) = \{A \subseteq \Theta : m(A) > 0\}$
$g(.)$	multi-label classifier, $g : \mathcal{X} \to 2^{\mathcal{Y}}$
$m(.)$	mass function or basic belief assignment, $m(.) \in [0,1]$
n	total number of examples in training data set S
N	number of subclassifiers
$P(c)$	prior probability of class c example
$P(c\|c')$	conditional probability of assigning label c to a class c' example
$\mathbb{R}$	set of all real numbers
$rank_f(x,.)$	rank of $f(x,.)$, one-to-one mapping from $\mathcal{Y}$ on to $\{1,\ldots,C\}$
S	set of training examples
x	an instance, $x \in \mathcal{X}$
$\mathcal{X}$	instance space of p-dimensional features
Y	set of class labels of the instance x, $Y \subseteq \mathcal{Y}$

$\mathcal{Y}$	finite set of labels or classes
$\|Y\|$	cardinality or "size" of a set Y
$Y(c)$	$Y(c) = 1$ if $c \in Y$; $Y(c) = 0$ otherwise
$\parallel a \parallel$	$\parallel a \parallel = 1$ if a is true; $\parallel a \parallel = 0$ if a is false
$2^{\mathcal{Y}}$	power set of $\mathcal{Y}$

$Avg\_Prec_S(f)$	average precision
BBA	basic belief assignment
BEP	break-even point
BIA	baseline induction algorithm
BoE	body of evidence
$Coverage_S(f)$	coverage
CV	cross validation
DECF	Dempster's evidence combination function
DST	Dempster-Shafer theory of evidence
F_β	F_β-measure
FN	false negatives, number of positive examples misclassified as negative
FoD	frame of discernment
FP	false positives, number of negative examples misclassified as positive
$HL_S(g)$	Hamming loss
k-NN	k-nearest neighbor algorithm

$One\_Err_S(f)$	one-error
PCA	principal component analysis
Pr	precision, $Pr = TP/(TP + FP)$
Re	recall, $Re = TP/(TP + FN)$
$RL_S(f)$	ranking loss
SVM	support vector machine
TN	true negatives, number of correctly classified negative examples
TP	true positives, number of correctly classified positive examples

List of Figures

List of Tables

CHAPTER 1

Introduction

Machine learning is a subfield of artificial intelligence that is concerned with the development of algorithms and techniques that make computers capable of acquiring skill and integrating knowledge from data or experience, with the objective of solving a given problem in a way analogical to human learning (Michalski et al. 1998). It can be viewed as a general inductive process that discovers the inherent structure in large bodies of data and builds a model in order to make useful decisions. An algorithm that automatically improves its performance through experience is said to have learned. Machine learning usually refers to the changes in systems that perform tasks involving recognition, diagnosis, planning, robot control, prediction, etc. If a system's task involves prediction, then improving performance means improving its ability to predict key elements of the task. The capacity to learn from experience or examples results in a system that can continuously self-improve and thereby offer increased efficiency and effectiveness. Machine learning is a rapidly expanding field with many successful applications in diverse disciplines such as bioinformatics, security informatics, Internet traffic routing, Internet search engines, information retrieval, data mining, signal and image processing, robot locomotion, autonomous driving, finance, economy, and business to name a few.

A particular task requiring machine learning technology that is of interest is classification. Classification has a broad spectrum of applications which includes medical diagnosis, biometric identification, speech and handwriting recognition, optical character recognition, object recognition in computer vision, credit scoring, detecting credit card fraud, automated stock trading, classifying DNA sequences, predicting protein structure, image classification, and natural language processing. The term "classification" sometimes is used to describe the learning process by which a classification system is constructed, but often it describes the result of such a process. Induction of classifiers is a common machine learning task. Classifiers are classification schemes or rules that are extracted from a set of training data that consists of examples described by attribute or feature vectors and class labels. The classification rules relate the class labels to some features. This work considers supervised learning where the features of examples are observable and classes of examples are pre-defined and known in the training set. Given a new instance or object whose features' values are observed, the typical classification problem is to determine the class of which the new instance is a member according to the classification scheme.

Traditionally the classification is concerned with assigning an object to exactly one class, which is called multi-class, single-label classification problem. However, many real applications are involved with the situation where each example can belong to several classes simultaneously. In other words, the sets of examples belonging to different classes are not disjoint. This is called multi-class, multi-label classification. The problem has been encountered in many areas, e.g., medical diagnosis where a patient may be suffering from more than one health condition: diabetes, high blood pressure, high cholesterol; book classification where a book is cataloged into multiple categories: engineering, technology, science, computers; film classification where a movie straddles several film genres: science fiction, horror, action, and adventure;

and scene analysis where an image belongs to many semantic classes: beach, ocean, sky, sunset, and mountain. The classification of text documents under multi-label setting is the problem that is contemplated in this dissertation.

1.1 Motivation

Information retrieval is the process of searching, sorting, recovering, and interpreting information from large amounts of stored data, which is crucial to documentation and organization of knowledge. Most information nowadays is generated and stored digitally. Business and personal correspondence, scientific and entertaining articles, conference proceedings, electronic journals and text archives, medical literature, patient data are just a few examples of electronic text collections. With the advent of World Wide Web (WWW) another massive repository of text information was created. The Internet has become the world's largest source of information, e.g., books and periodicals, dictionaries, thesauri, digital documents collected in a wide range of subject domains can now be found online. These huge text data demand automatic means of efficient and effective storage and retrieval. Classification is an important task of information retrieval systems. Given a collection of documents, one purpose of the system is to find and retrieve categories of relevant documents, and this can be provided through text categorization.

Text categorization is a necessity in information retrieval. It offers tools for converting unstructured text collections into structured one such that storage and search get easier and faster. For instance, text categorization techniques can be applied to unstructured documents such as call center logs or email messages that an enterprise receives to help them understand issues/complaints that their customers are having with their products or services. This will enhance and provide more intelligence to their enterprise's search capabilities. Specifically, text categorization handles and organizes text data with the goal of assigning one or more category labels from a pre-defined set to a document. Several applications of this document processing task include cataloging of new books following library of congress subject headings, indexing of scientific articles according to thesauri of technical terms, keyword tagging, authorship identification, automatic content management, and information filtering and routing such as filing patents into patent directories. The tremendous volume of information available online and its rapidly continuing growth also make it necessary to use text categorization techniques in Web mining and email management. Some examples are the classification of new stories or Web resources, the identification of document genre, personalized routing of news articles, the search of information on the WWW through the hypertext, filtering of unwanted content for Internet browsers, email archiving, spam filtering in email messages, and many others.

Application domains of multi-label classification, particularly text categorization, are numerous and important, and they are bound to increase dramatically in both number and importance due to advances in computer technology allowing the proliferation and acquisition of documents in digital form. One example of a real-life document collection is PubMed, an online service of the U.S. National Library of Medicine, which includes over 17 million citations from MEDLINE and other life science journals for biomedical articles back to the 1950s [1]. There are an estimated 40,000 new biomedical abstracts being added monthly to this collection (Feldman and Sanger 2007). Thus one can anticipate an immense potential of multi-label classification techniques in the future. Some neighboring fields of text categorization where methods and techniques of multi-label classification are relevant and can be extended to are the classification of very noisy text resulting from optical character recognition

[1]from `http://www.pubmed.gov`, accessed on 07-29-2007

(Ittner et al. 1995), the classification of speech transcripts (Schapire and Singer 2000), image annotation and labeling (Johnson and Cipolla 2005), and retrieval of content-specific and high quality articles in internal medicine for the purpose of providing patients the best customized care (Aphinyanaphongs et al. 2005). Other applications include emotion detection in music (Li and Ogihara 2003), semantic scene classification (Shen et al. 2004), prediction of genes' functional classes (Clare and King 2001; Zhang and Zhou 2006), and protein subcellular localization identification (Su et al. 2005).

The categorization process is usually time-consuming and costly. Due to the ever-increasing number of documents, it is beyond human's ability to manually organize vast amounts of material in a reasonable time frame with a fair amount of effort, even by trained professionals. As a result, there is a growing need for tools to solve multi-label classification problems in order to help people better manage resources and improve the accuracy and efficiency of categorization. However, so far the subject has received insufficient attention.

1.2 Challenges and Research Objective

The task of inducing multi-label or text classifiers poses a number of challenges and difficulties which arise from some intrinsic complexity of the problems:

- Input data space of potential examples is extremely large. This is often the case for text classification problems where data sets consist of hundreds of thousands of documents that are characterized by tens of thousands of terms or features and are to be categorized into one or more of thousand categories. Machine learning methods are computation intensive. Training classifiers in a high-dimensional feature space with several thousands of training examples will be a major computational burden for conventional learning techniques; the work undertaken could exceed the capacity of available computing resources.

- It is not unusual to find that many of the available features are not informative and have little discriminative power to predict category membership. These features are considered redundant or irrelevant to the problem. Yet each feature used still adds to the running time and costs of a classification system. So, for interpretability and economical reason, decision rules that rely on a small subset of features are desirable. In addition, it has been shown that including irrelevant features often reduces a classifier's accuracy, thus more is not always better. Using too many features can yield inaccurate classification just as bad as using an insufficient number of features. Lewis (1992b) examined the effect of feature set size on text categorization effectiveness and observed the decrease in effectiveness with increasing feature set size; for word-based indexing, the small number of features (about 10 to 15 features) was found to be optimal. Yang and Pedersen (1997) discussed feature space reduction and evaluated some feature selection methods. It is critical to select features that are necessary and sufficient to represent a problem space; each individual feature should clearly capture some aspects of the problem space (Blake and Pratt 2001).

- The dimension of the feature space is highly likely to be much larger than the number of training examples. This is known as curse of dimensionality, which is a significant obstacle in machine learning problems that involve learning from a low number of examples in a high-dimensional feature space. It causes difficulties when fitting a model or performing some numerical computations.

- The data space is very sparse. Typically, most documents contain fewer words in comparison to the total number of words in the entire document collection.

Therefore, when using words as features, text documents are usually represented as high dimensional and sparse vectors. Because of sparsity, certain patterns would happen infrequently in data making it difficult to learn or extract any rules from them to make intelligent decisions. Learning in high dimensions with sparse problem space was explored by Lafferty and Wasserman (2006).

- Multi-label examples are often highly imbalanced; some of the classes are heavily populated while some of the classes only contain a few examples. Studies have shown that imbalanced training data can adversely affect classification accuracy of a classifier (Liu and Motoda 2002).
- The process of generating document data sometimes creates mislabeled and noisy examples. Typographical errors, spelling errors, abbreviations, and variations in the words/terms used in documents are some sources of noise in data. Noisy information and incorrect labeling in the training examples would mislead the classifier induction and prevent it from discovering important inherent patterns.

Among the many challenges of multi-label classification problems mentioned above, the dissertation aims to address one particular facet which is the induction of classifiers when the input data space is large. To be specific, the author seeks to develop an algorithm to learn from multi-labeled examples that are described by a large number of features and to generate classification rules for future categorization. Although the focus is on solving multi-label problems with a massive data space, the data sets that are used in the experiments to study the performance of the proposed method are of typical real-world, large-scale data sets which also posses other interesting characteristics such as sparsity, considerably more features than the total number of examples, imbalanced examples, and uninformative features.

Numerous classification methods exist in literature. However, some methods are not appropriate for practical use, e.g., they require the knowledge of the number of labels of a new example in order to categorize, which is often unrealistic. Some of them perform better than others in terms of having higher predictive accuracy, and some may have other nicer qualities such as robustness, computational simplicity, interpretable classification models, and the potential to handle feature spaces of high dimensions. Though, frequently a practical consideration in choosing one method over the others is the computational cost which includes the training time and classification time. For an extremely large training corpus, this cost becomes a big obstacle that may not be overlooked, e.g., training is excessively time consuming. Importantly, conventional classification methods applied to a large data set can take several days to complete the classification task, which would be formidable.

Thus, to perform text classification in practice, there is a need for an alternative learning mechanism that is able to handle a vast amount of data cost-effectively. The goal here is to devise a technique that performs well and robustly for real world problems and that is computationally efficient. The author approaches the inductive learning of text classifiers by means of the fusion of multiple multi-label classifiers.

1.3 Summary of Contributions

The time required for induction of many learning methods grows quickly, sometimes exponentially, as a function of the number of features (Vafaie and Jong 1994; Koller and Sahami 1996). The experiments in the early phase of this study reached the same finding as well. For instance, using 10,000 features incurred much more than ten times the CPU time that was needed when using 1,000 features. In general, if there are N feature sets and T_j is the time needed to induce a classifier from the

j-th feature set, then the total time to induce N classifiers, one from each of the N feature sets, is $\sum_{j=1}^{N} T_j$ and $\sum_{j=1}^{N} T_j \ll T$, where T is the time needed to induce a classifier from all features in the N feature sets combined. This observation suggests a viable resolution for the induction of classification rules from the training data set of multi-labeled examples with a very high dimensional feature space. Rather than working on the whole set of a large number of features, the computational difficulty is reduced by partitioning the big feature set into many smaller feature subsets where each small feature subset is used to construct a subclassifier, and then outcomes of multiple subclassifiers are subsequently combined to produce a final classification prediction. This approach not only cuts down the overall computation time but also demands less computing resources during each induction of a subclassifier. Moreover, the induction of subclassifiers can be carried out concurrently, which will speed up the classification process even more.

To sum up, the proposed classifier induction system executes two basic tasks, inducing subclassifiers from different feature subsets and fusing subclassifiers' outputs. The hypothesis put forth in this dissertation is that the proposed schema reduces substantially the computational overhead for learning while the final classification performance does not deteriorate materially. In order to achieve this, the system must have a good learning method as a baseline algorithm for inducing subclassifiers and a good fusion technique for combining classification results. There are many multi-label learning methods that may be selected as candidates for a baseline induction algorithm. For a master algorithm that combines multiple classifiers, several methods are also available, though they are mostly applied to single-label classification and must be modified for multi-label case. The choice of a fusion method has a central role to play in this classification arrangement since one must make sure that for an exchange in the savings in time, this scheme gives an acceptable accuracy that is not much worse than the usual classification based on all features. Adopting the Dempster-Shafer evidence combination method, the author developed a new algorithm that fuses outputs of a set of subclassifiers for multi-label problems. The existing weighted sum classifier combination method was also modified such that it can handle multi-label classifiers. The experiments indicated that these two methods performed comparably and were better than other voting-based combination methods. However, fusion always led to a little accuracy loss as compared to no-fusion, all-features classification, while the processing time was significantly reduced.

Feature selection is a crucial step in successful text categorization. Its objectives are to provide a better understanding in data, to lower the costs of data management and processing, and to improve the classification performance. Feature selection has been extensively researched; some literature includes Lewis (1992b), Yang and Pedersen (1997), Guyon and Elisseeff (2003), and Forman (2007). In this work, the issues regarding how features in the original feature set are selected were left aside. It was assumed that some feature selection method was implemented before proceeding with this classification scheme. Nevertheless, the question about how the feature set should be divided into subsets to obtain the most favorable results was explored briefly. Specifically, a concern related to the effect of breaking up the mutual relationship among features when the feature set is partitioned was investigated. Via experiments it was shown that placing the features that are interdependent in the same feature subset instead of a random assignation made no difference in categorization accuracy. In determining the linear dependency structure between features, the author opted for a multivariate statistical technique called principal component analysis. Furthermore, there was no concrete evidence to support the use of other more complicated ensemble generation techniques. Results obtained were rather consistent that for the proposed fusion method, the random assignment of features worked reasonably well.

1.4 Outline of the Dissertation

The organization of this dissertation is as follows. Chapter 2 begins with a brief overview of multi-label classification problems that introduces the problem statement and general approaches to solving the problems. The performance criteria used in evaluating this type of problems are reviewed. Chapter 3 surveys related work in the areas of multi-label classification and text categorization. This includes the discussion of some multi-label classification methods that are used in the experiments, two boosting-based algorithms, the AdaBoost.MH and alternating decision tree algorithms, and two other single-label methods that are adapted for multi-label problems, C4.5 and k-nearest neighbor.

Presented in Chapter 4 are the results from a preliminary study that applied these classification methods to a large real-world database, EUROVOC thesaurus. These results led to the search for a better solution to the problem. The proposed methodology, the fusion of multiple multi-label classifiers, is explained, and the choice of a baseline induction algorithm is examined experimentally. Chapter 5 first gives a short review of various classifier combination techniques and basic concepts of the Dempster-Shafer theory of evidence that provides the foundation for the master algorithm. It then shows how the proposed Dempster-Shafer fusion method is developed for multi-label problems.

The performance of the Dempster-Shafer fusion is demonstrated through experiments on two data sets, EUROVOC thesaurus and Reuters corpus. Experimental results are reported in Chapter 6. Chapter 7 compares Dempster-Shafer fusion to other classifier combination techniques. After that, Chapter 8 considers how feature subsets should be created. Lastly, Chapter 9 concludes the dissertation and discusses possible directions for future research work.

CHAPTER 2

Multi-label Classification

The traditional classification involves the assignment of a particular class in a set of pre-defined classes to an instance or object as per the classifier that was induced from a given set of training data. This class membership prediction is single-label classification and known as multi-class classification or binary classification, if there are only two classes. In multi-class, multi-label problems, however, each instance is associated with a set of labels or classes. That is, the event that an instance belongs to a class A and the event that it also belongs to another class B are not mutually exclusive as in the single-label problem. Multi-label classification has found applications in various fields such as bioinformatics, emotion detection in music, film categorization, semantic scene classification, text mining, and document classification. For example, in text categorization, a text document may belong to many subjects or topics; in bioinformatics, one protein may have many effects on a cell; in music categorization, a piece of music shares many musical genres. The classification task of these cases is to output a set of labels whose size is not known in advance; this may vary from the assignment of a single category label to many different labels. One document may, for instance, talk about business, investment, and finance, although another one would concern only business and marketing. One image may show a beach scene at sunset with a mountain background, while another image shows fall foliage of a mountain and the blue sky.

This chapter provides a formal statement of a multi-label classification problem and some relevant basic definitions. Two general approaches in dealing with the problem and how to measure the performance of a classifier will be discussed.

2.1 Problem Statement

Let $\mathcal{X} \subseteq \mathbb{R}^p$ be an instance space of p-dimensional features and $\mathcal{Y}$ be a finite set of labels or classes. Each instance $x \in \mathcal{X}$ has multiple class labels in Y, where $Y \subseteq \mathcal{Y}$. Given a set of training examples consisting of n instances, $S = \{(x_1, Y_1), \ldots, (x_n, Y_n)\}$, where each instance is independent and identically distributed (i.i.d.) drawn from an unknown distribution D, $x_i \in \mathcal{X}$ and labels $Y_i \subseteq \mathcal{Y}$ are known, the goal of multi-label classification is to learn categories' properties from labeled examples and find a multi-label classifier $g : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$ that maps an instance x to its label such that specific performance criteria are optimized. Here, $2^{\mathcal{Y}}$ is the power set of $\mathcal{Y}$, which is the set of all subsets of $\mathcal{Y}$.

Many machine learning algorithms induce decision rules in the form of ranking function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$. For a given instance x, the labels in $\mathcal{Y}$ can be ordered according to the scores or values of $f(x, .)$. This ranking function $f(x, c)$ is thus interpreted as the system's confidence that x belongs to class c. It is said that label c_1 is ranked higher than c_2 if and only if $f(x, c_1) > f(x, c_2)$. If Y is the set of class labels of x, then a good learning algorithm will rank labels in Y higher than those not in Y. A simple classification function, $g(x)$, can be easily obtained from the function $f(x, c)$ and a threshold, $t(x)$, by labeling x with the classes whose ranking values exceed $t(x)$:

$$g(x) \;=\; \{c | f(x,c) > t(x), c \in \mathcal{Y}\}. \tag{2.1}$$

The threshold $t(x)$ is usually chosen to be a constant function. The most common threshold is $t(x) = 0$.

Classifier's performance depends greatly on the characteristics of the data to be classified. One factor that has a certain bearing on the behavior of a multi-label classifier induced from a data set is the number of labels of each example in the set. In some applications, the number of labels each example has is large compared to the total number of labels in the entire data set, whereas in others it is small. Tsoumakas and Katakis (2007) introduced two pertinent concepts, *label cardinality* and *label density*, that can be used to describe a data set. Denote the cardinality or "size" of a set Y by $|Y|$. Here are the definitions of these two notions:

- Label cardinality of data set S is the average number of labels of the examples in S,
$$LC(S) \;=\; \frac{1}{n}\sum_{i=1}^{n} |Y_i|. \tag{2.2}$$
- Label density of data set S is the average proportion of labels of each example,
$$LD(S) \;=\; \frac{1}{n}\sum_{i=1}^{n} \frac{|Y_i|}{|\mathcal{Y}|}. \tag{2.3}$$

Both metrics quantify the number of alternative labels that depict the examples of a multi-label data set. Label cardinality is independent of the total number of labels in the classification problem, while label density takes the total number into account. Two data sets having the same label cardinality but different label density might exhibit different properties that influence the performance of the multi-label classification methods.

2.2 Solving Multi-label Problems

Existing multi-label classification methods may be grouped into two main approaches: 1) problem transformation and 2) algorithm adaptation (Tsoumakas and Katakis 2007). Problem transformation approach comprises those techniques that transform the multi-label classification problem into one or more conventional single-label classification problems, and algorithm adaptation approach comprises any methods that extend specific learning algorithms in order to cope with multi-labeled examples directly.

There are a number of ways that one could do to follow the problem transformation avenue.

1. Ignore multi-label information. Two straightforward methods are (Boutell et al. 2004):

   - Randomly or subjectively select one of the multiple labels of each multi-labeled example and dispose of the rest.
   - Simply remove every multi-labeled example from the data set.

   Obviously these methods have a serious shortcoming of discarding a lot of information content of the original multi-label data set.

2. Consider each different set of labels that exists in the multi-label data set as a single label and then learn one single-label classifier (Boutell et al. 2004; Diplaris et al. 2005). One negative aspect of this method is that it may lead to data sets with a large number of classes and few examples per class.

3. Learn $|\mathcal{Y}|$ binary classifiers, one for each different class label c in $\mathcal{Y}$ (Boutell et al. 2004; Lauser and Hotho 2003). The original data set is transformed into $|\mathcal{Y}|$ data sets such that the c^{th} data set, $c = 1, \ldots, |\mathcal{Y}|$, contains all examples of

the original data set, but each example is relabeled as c if the original example has label c, and $\bar{c}$ (not c) otherwise. When categorizing a new instance x, this method gathers and assigns to the new instance the labels output by all binary classifiers. Transforming a multi-label problem into a number of binary classification problems is the most common method of the problem transformation approach. However, this method has some disadvantages (Kang et al. 2006).

- It treats each class label independently, and therefore does not exploit any correlation among class labels.
- It does not scale very well to a large number of classes since a binary classifier has to be built for every class.
- It suffers severely from the unbalanced data problem, particularly when the number of classes is large. This is because for the class that has a few examples, say class c, examples having label $\bar{c}$ vastly outnumber examples having label c. Consequently, it is likely that the binary classifier will output the label $\bar{c}$ for all instances.

4. Decompose each example (x, Y) into $|Y|$ examples (x, c) for all $c \in Y$. Then induce one single-label classifier from the transformed data set using a learning method that can give to an instance certainty degrees or probabilities for all labels in $\mathcal{Y}$. A new instance is assigned those labels whose certain degrees are greater than a set threshold. This method increases the size of training examples, which is a downside.

Rather than transforming data or changing the nature of multi-label problems, researchers who take up the algorithm adaptation approach seek to modify single-label classification methods such that they can handle multi-labeled examples straight out. There are a number of algorithms in this group. Since our proposed methodology relies on some methods in algorithm adaptation category, a full length discussion on these methods is deferred to the next chapter.

Apart from the choice of classification methods, there are essentially two different types of categorization decision that one can make, *hard* categorization and *soft* (or ranking) categorization (Sebastiani 2006; Feldman and Sanger 2007). A hard categorization decision refers to a binary decision, yes or no, as to whether an instance x belongs to a category c. A soft categorization decision is the one consisting of attributing a real-valued score or weight to the example-category pair (x, c) indicating goodness-of-fit between the input document and the category. This score reflects the degree of confidence of the classifier in the fact that x belongs to category c, which allows ranking a set of categories in terms of their appropriateness for x, or ranking a set of instances in terms of their appropriateness for category c. A disadvantage of soft categorization or a ranking algorithm is that it does not output a set of labels for the example. A group of top scoring categories or all the categories with the scores above a chosen threshold may be selected. However, the threshold parameter will need tuning in each problem. The two classification functions, $g(x)$ and $f(x, c)$, described in Section 2.1 above are used in hard and soft decisions, respectively. Some classification methods offer the convenience of making both soft and hard decisions, but some methods only permit one decision type.

2.3 Performance Criteria

Different classification tasks are needed for different applications since no single classifier is suitable for all problems. An important issue when doing classification is measuring the performance of the classification method, so that one knows how good a classifier is and can make a choice for an appropriate method. Normally one would be interested in the effectiveness and the efficiency of classification methods. The effectiveness refers to the ability of the classifier to take the right classification decision, e.g., having high accuracy, and the efficiency refers to the time to perform a task, e.g., the training efficiency is the average time it takes to build a classifier from a training set and the classification efficiency is the average time it takes to classify previously unseen examples. The evaluation is typically conducted experimentally using some independent test data sets or via cross validation.

Assigning the label of a class of interest to examples is merely a binary classification, which is the simplest classification problem where an example is classified as either belonging to or not belonging to the category. In particular, for a category c, examples are organized into two groups: the group of "positive" examples that are labeled by c, and the group of "negative" examples that are not labeled by c. The classification outcomes are usually tabulated in a confusion matrix or a two-way contingency table as shown in Table 2.1. The four quantities displayed in the table are:

1. TP (true positives), the number of correctly classified positive examples.
2. FN (false negatives), the number of positive examples misclassified as negative ones.
3. FP (false positives), the number of negative examples misclassified as positive ones.
4. TN (true negatives), the number of correctly classified negative examples.

Many measures of effectiveness have been proposed; each of which is designed to evaluate some aspects of the classification performance of a method. The standard evaluation measures that are of the most widespread use in domains where each

Table 2.1: A contingency table showing classification results for a category c

		True category	
		c	Not c
Classification outcome	c	TP	FP
	Not c	FN	TN

example belongs to a single class are error rate, precision (or positive predictive value), recall (or sensitivity), and F-measure (Aas and Eikvil 1999; Yang 1999). The followings define the first three measures:

- *Error rate* (Err) is the rate of incorrect assignments.

$$Err = \frac{FP + FN}{n} \tag{2.4}$$

  Here, $n = TP + FP + FN + TN > 0$ is the total number of examples.

- *Precision* (Pr) is the rate of correct assignments when the examples are assigned to category c.

$$Pr = \frac{TP}{TP + FP}, \quad \text{if } TP + FP > 0; \quad \text{otherwise undefined.} \tag{2.5}$$

- *Recall* (Re) is the rate of correct assignments for examples in category c.

$$Re = \frac{TP}{TP + FN}, \quad \text{if } TP + FN > 0; \quad \text{otherwise undefined.} \tag{2.6}$$

Usually error rate is not a good performance measure in text categorization (Joachims 2003). Due to the number of negative examples being overwhelmingly larger than the number of positive examples in general, any classifier that always categorizes an instance as negative regardless of its feature values will have a somewhat misleading low error rate.

Interpreting precision and recall also requires caution. Neither precision nor recall can effectively measure classification performance on its own; they are often deceptive when examined alone (Sebastiani 2002; Yang 1999). Generally, a classifier exhibits a trade-off between these two measures. A higher precision (or recall) can be obtained by adjusting the internal parameters or changing the decision threshold of the classifier. However, getting a high precision usually means sacrificing recall and vice versa. If the recall and precision can be tuned to have an equal value, then this value is the break-even point of the classifier. Break-even point (BEP) was first proposed in 1992 by Lewis (1992a), but, as quoted in (Sebastiani 2002), was criticized by himself in 1997 that BEP is not a good effectiveness measure. It is possible in some cases that BEP is not obtainable. For instance, when there are only a few category c test examples compared to a large number of examples not in category c, the recall value can be so high that the precision will never reach (Sun and Lim 2001). Furthermore, to have precision equal recall is not necessarily desirable, and it is not clear that a classifier that achieves high break-even point will score high on other effectiveness measures.

While precision and recall are useful measures, it is easier to compare different learning algorithms by considering one score instead of two scores. Observing that one usually wants to maximize both precision and recall, at the same time balancing their values, van Rijsbergen (1979) proposed combining them into a single score, F_β-measure, controlled by a user-defined parameter β, $\beta \in [0, \infty)$, that quantifies the relative degree of importance ascribed to either precision or recall:

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re} \tag{2.7}$$

It can be computed directly from the contingency table using

$$F_\beta = \frac{(\beta^2 + 1)TP}{(\beta^2 + 1)TP + FP + \beta^2 FN} \tag{2.8}$$

F_β is simply the weighted harmonic mean of precision and recall. Value of β controls the trade-off between the two measures. $\beta > 1$ gives more weight to recall and $\beta < 1$ gives more weight to precision. F_β converges to recall when $\beta \to \infty$ and to precision when $\beta = 0$. Normally, $\beta = 1$ is used (Yang 1999). When $\beta = 1$, precision and recall are deemed equally important, in which case F_1 degenerates to the following form:

$$F_1 = \frac{2 \times Pr \times Re}{Pr + Re}. \tag{2.9}$$

or

$$F_1 = \frac{2TP}{2TP + FP + FN}. \tag{2.10}$$

Yang (1999) showed that F_1's score is maximized when the values of recall and precision are equal, i.e., $F_1 = Pr = Re$; otherwise, the smaller of recall and precision dominates the value of F_1.

Two other commonly used F_β-measures are the $F_{0.5}$-measure, which weights precision twice as much as recall, and the F_2-measure, which weights recall twice as much as precision.

As one usually wants to know how well every label can be predicted, it is preferable to have a single measure that indicates the average performance of a classifier over all labels. Based on the foregoing preliminaries, Yang and Liu (1999) and Yang (1999) proposed two alternative ways to generalize these criteria to the needs of induction from multi-labeled examples: (1) *macro-averaging*, where precision and recall are first computed for each category separately, and then averaged over the resulting performance measures of different categories using arithmetic mean; and (2) *micro-averaging*, where precision and recall are computed from the averaged contingency table or, more conveniently, from the global contingency table obtained by summing over all individual decisions of every category. The corresponding formulas of both versions are derived as follows.

Suppose there are C categories in the category set $\mathcal{Y}$, i.e., $|\mathcal{Y}| = C$. The classification outcomes of a category c can be summarized as in Table 2.2. Aggregating results of all categories renders the global contingency table in Table 2.3.

Table 2.2: Contingency table for a category c

Category c		True category	
		c	Not c
Classification outcome	c	TP_c	FP_c
	Not c	FN_c	TN_c

Table 2.3: Global contingency table

Category set $\mathcal{Y}$= $\{1, ..., C\}$		True category	
		Yes	No
Classification outcome	Yes	$\sum_{c=1}^{C} TP_c$	$\sum_{c=1}^{C} FP_c$
	No	$\sum_{c=1}^{C} FN_c$	$\sum_{c=1}^{C} TN_c$

From Table 2.2, precision and recall of the classifier as measured on the c-th class label are computed as

$$Pr_c = \frac{TP_c}{TP_c + FP_c} \tag{2.11}$$

$$Re_c = \frac{TP_c}{TP_c + FN_c}. \tag{2.12}$$

Averaging precision and recall of all categories gives macro-average precision, Pr^M, and macro-average recall, Re^M, shown in Table 2.4. For micro-averaging method, Equations 2.5 and 2.6 are applied to the collective classification results in Table 2.3 to give micro-average precision, Pr^μ, and micro-average recall, Re^μ, in Table 2.4.

Table 2.4: Macro-averaging and micro-averaging versions of precision, recall, and F_1-measure for domains with multi-labeled examples.

	Macro (M)	Micro (μ)
Precision	$Pr^M = \frac{\sum_{c=1}^{C} Pr_c}{C}$	$Pr^\mu = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c+FP_c)}$
Recall	$Re^M = \frac{\sum_{c=1}^{C} Re_c}{C}$	$Re^\mu = \frac{\sum_{c=1}^{C} TP_c}{\sum_{c=1}^{C}(TP_c+FN_c)}$
F_1	$F_1^M = \frac{\sum_{c=1}^{C} F_{1,c}}{C}$	$F_1^\mu = \frac{2\times Pr^\mu \times Re^\mu}{Pr^\mu+Re^\mu}$

Also shown in Table 2.4 are the macro-average and micro-average F_1-measures. The $F_{1,c}$ in the formula of the macro-average F_1, F_1^M, is the F_1-measure pertaining to the classifier for the class c:

$$F_{1,c} = \frac{2 \times Pr_c \times Re_c}{Pr_c + Re_c}. \tag{2.13}$$

Macro-averaging weights equally all the categories regardless of how many examples each category contains, whereas micro-averaging weights equally all the examples. That is, macro-averaged performance measure is considered a per-category average, and micro-averaged measure is considered a per-example average. The two methods may give quite different results, especially if different categories have very different number of examples or the distribution of training examples across the categories is highly skewed (Sebastiani 2002). Normally smaller classes tend to be harder to classify and often there are more of them than larger classes. According to Yang and Liu (1999), the micro-averaging score is more influenced by the classification performance on common or large classes and the classification performance on rare or small classes controls the macro-averaging score. As an example, a good performance of a classifier on common classes will result in a high micro-average F_1 score.

As argued by Shen et al. (2004), allowing each example to belong to more than one class means that the classification can be either fully correct, or correct to a certain extent, or fully wrong. The classical performance criteria, however, do not reflect this circumstance. Five more appropriate evaluation measures have been suggested; most of them are customized to reflect certain aspects of the classifier's performance in ranking (Schapire and Singer 2000).

For an example x, assume that a ranking function f returns a value, $f(x, c)$, for each class label c. Denote the rank of $f(x, c)$ by $rank_f(x, c)$. That is, $rank_f(x, c)$ is a one-to-one mapping on to $\{1, \ldots, C\}$ such that if $f(x, c_1) > f(x, c_2)$, then $rank_f(x, c_1) < rank_f(x, c_2)$. Given the training set denoted by S, the multi-label performance criteria are defined as follows (Zhang and Zhou 2005):

- *One-error* measures the probability that the top-ranked label was not in the set of possible labels. If we will assign a single label to an example, then one-error measures the probability of not getting even one of the labels correct. It is computed as the proportion of the number of times that the top-ranked label fails to appear in the set of correct labels to the total number of examples.

$$One\_Err_S(f) = \frac{1}{n} \sum_{i=1}^{n} \parallel argmax_{c \in \mathcal{Y}} f(x_i, c) \notin Y_i \parallel \tag{2.14}$$

  where

$$\parallel a \parallel = \begin{cases} 1, & \text{if } a \text{ is true,} \\ 0, & \text{if } a \text{ is false.} \end{cases}$$

  The smaller the value of $One\_Err_S(f)$, the higher the performance. In domains where $|Y_i| = 1$ for each x_i, i.e., single-label classification problems, $One\_Err_S(f)$ is identical to ordinary classification error.

- *Coverage* measures how far on average one needs to go down the list of labels assigned by the classifier in order to detect all the labels of an example.

$$Coverage_S(f) = \frac{1}{n} \sum_{i=1}^{n} \max_{c \in Y_i} rank_f(x_i, c) - 1 \tag{2.15}$$

The smaller the value of $Coverage_S(f)$, the higher the performance. In domains where $|Y_i| = 1$ for each x_i, $Coverage_S(f)$ corresponds to the average rank of the correct labels, and equals zero if the classifier does not commit any classification errors.

- *Average precision* measures the effectiveness of the label rankings. It is computed as the average fraction of those labels from Y_i that are ranked above a particular label $c \in Y_i$.

$$Avg\_Prec_S(f) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{|\mathrm{Y}_i|} P(x_i) \tag{2.16}$$

where

$$P(x_i) = \sum_{c \in Y_i} \frac{|\{c' \in Y_i | rank_f(x_i, c') \leq rank_f(x_i, c)\}|}{rank_f(x_i, c)} \tag{2.17}$$

The higher the value of $Avg\_Prec_S(f)$, the higher the performance, the maximum value being $Avg\_Prec_S(f) = 1$. When $Avg\_Prec_S(f) = 1$, the classifier achieves the perfect performance; no instance x_i for which a label not in Y_i is ranked higher than a label in Y_i.

- *Ranking loss* is the average fraction of crucial pairs which are misordered. A crucial pair is the pair of labels c_0, c_1 for which $c_0 \notin Y$ and $c_1 \in Y$. A classification rule f misorders a crucial pair c_0, c_1 if $f(x, c_1) \leq f(x, c_0)$, i.e., f fails to rank c_1 above c_0.

$$RL_S(f) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{|Y_i||\mathcal{Y} - Y_i|} |\{(c_0, c_1) \in (\mathcal{Y} - Y_i) \times Y_i \mid f(x_i, c_1) \leq f(x_i, c_0)\}| \tag{2.18}$$

The smaller the value of $RL_S(f)$, the better performance.

- *Hamming loss* is the fraction of samples and labels that are incorrectly identified.

$$HL_S(g) = \tfrac{1}{Cn} \textstyle\sum_{i,c} (\| c \in g(x_i) \wedge c \notin Y_i \| + \| c \notin g(x_i) \wedge c \in Y_i \|) \tag{2.19}$$

Here, $g : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$ is a multi-label classifier as defined by Equation (2.1).

The smaller the value of $HL_S(g)$, the higher the performance. When $|Y_i| = 1$ for each x_i, *Hamming loss* is simply $\frac{2}{C}$ times the loss of the usual classification error.

CHAPTER 3

Literature Review

The application of multi-label classification is found mostly in text mining, and as such the literature in this subject is primarily directed to text categorization. As indicated in Section 2.2, the two common approaches to multi-label classification problems are problem transformation and algorithm adaptation. Majority of research work fall into the former group with a huge bibliography of learning algorithms available, and much less are the methods in the latter group.

This chapter gives a summary of work related to multi-label classification with emphasis on methods for text categorization. Details will be provided on four specific methods that are used in the experiments. Two of them, AdaBoost.MH and ADTree (alternating decision tree), are multi-label learning algorithms, and the other two, C4.5 and k-nearest neighbor algorithm, are by and large for single-label learning.

3.1 Related Work on Multi-label Classification

A simple approach to a multi-label classification problem is to formulate the problem as multiple binary classification problems for which one binary classifier is to be built for one category (Yang 1999; Joachims 1998; Nigam et al. 2000). That is, each category has its own classifier separate from other categories which is used to determine if an example belongs to that category. Collective decisions made by every binary classifier produce a set of category labels for the example. This is accomplished by defining a score function that computes a real-valued score indicating goodness-of-fit between the input example and the category. A group of top scoring categories or all the categories with the scores above the given threshold will be selected.

An advantage of this approach is that binary classifiers can be conveniently generated by applying any available statistical classification methods or machine learning techniques. One method that has been found to perform favorably is Bayesian classifier (Langley et al. 1992; Friedman et al. 1997; Eyheramendy et al. 2003). McCallum and Nigam (1998) compared two first-order probabilistic classifiers that adopted the naive Bayes assumption: the multinomial model and the multivariate Bernoulli model. In terms of classification performance, the multinomial model was almost uniformly better than the multivariate Bernoulli model. Vilar et al. (2004) also presented satisfying results on using the multinomial (Naive Bayes) classifier on the Reuters-21578 data set. In spite of producing good outcomes, probabilistic models suffer some criticisms: 1) the exact form of the joint distribution is usually unknown in real-world problems and 2) the naive Bayes assumption that features must be independent of one another is unrealistic.

Other methods under this approach include neural networks, k-nearest neighbor (k-NN) algorithm, and support vector machines (SVM). Ruiz and Srinivasan (1998) presented the results obtained from a series of experiments in automatic text categorization of MEDLINE articles. They implemented counterpropagation neural networks and trained them to assign categories based on term frequency of single words from title and abstract. However, they reported that their networks performed worse than backpropagation network. k-NN is a popular supervised learning algorithm for single-label classification that often achieves very good performance when applying to text categorization problem. Han et al. (2001) showed that their weight adjusted k-NN algorithm outperformed Naive-Bayes classification techniques, neural networks, and decision tree induction algorithm like C4.5. Calvo et al. (2004) implemented the Naive Bayes classifier for web applications using an object oriented classification framework, and compared it to k-NN classifier. Both methods were comparable in terms of accuracy, but k-NN was not suited for applications that required classification to be performed at real time. Two main drawbacks of the k-NN technique are its computational inefficiency and its sensitivity to the choice of parameter k. k-NN is not a suitable method for highly imbalanced data; the classes with the more frequent examples tend to dominate the categorization of a new example. Rather than using a fixed k, Li et al. (2004) proposed a revised k-NN algorithm that uses different k values for different classes in accordance with their distribution in the training set in order to reduce the bias toward heavily populated classes.

Joachims (1998) and Kwok (1998) showed that SVM is appropriate for learning text classifiers with many nice properties. SVM is good at handling domains where the number of features exceeds the number of training examples. It eliminates the need for feature selection, saving a complicated preprocessing step. It is well suited for problems with few irrelevant features and sparse example vectors, and it behaves robustly without the need for manual parameter tuning. Ease in incorporating new examples, when available, into an existing trained system is also a plus. In their experiments, SVM substantially and significantly outperformed other text classification methods. On the downside, SVM induction is computationally more expensive than that of naive Bayes and k-NN, but roughly comparable to the C4.5 decision tree algorithm. At classification time, SVM is faster than k-NN, however. An improvement to standard SVM proposed by Joachims (2006) is a Cutting-Plane Algorithm called

SVM-Perf. Compared to existing methods, SVM-Perf is simple and easy to implement. It is faster than SVM that uses decomposition methods, and in his experiments there was no indication that SVM-Perf was less accurate.

There are two questions related to the use of multiple binary classification approach: how many labels to be assigned to a document and how to select the values of thresholds. Nevertheless, another important issue which should be of more concern is the fact that approaches that induce each binary classifier independently tacitly ignore the mutual relations among different classes. When one label provides information about another, binary classifiers fail to capture this. One simple way to take the correlations into consideration is to treat each possible combination of classes as a new class, and transform the multi-label problem into a usual single-label classification problem. Yet this may not be possible to do when there are a large number of classes. Moreover this causes the data space to become too sparse.

Instead of training binary classifiers independently, McCallum (1999) employed Bayesian classification approach in which the multiple classes that comprise a document are characterized by a probabilistic generative model that represents the correlations between class labels. Each different set of labels is considered independently as a new class, and the parameters of the model are learned by maximum a posteriori estimation from labeled training documents. Given a new document, the label set that is most likely is selected with Bayes rule. There are no thresholds to be tuned or learned. McCallum found that the mixture model significantly reduced classification error compared to the approach based on a group of binary classifiers.

BoosTexter by Schapire and Singer (2000) is a collection of enhancements to AdaBoost (Freund and Schapire 1997) that enables its application to multi-class, multi-label document classification problems. BoosTexter aims to predict all the correct labels by ranking them so that the correct labels receive the highest rank. To decide which labels to assign to a document, a threshold must be specified for the ranking function, and it is typically set as 0. Schapire and Singer did not evaluate any method of tuning the threshold on the ranks. Although BoosTexter can be seen as an algorithm adaptation method, at its core, it actually takes on the problem transformation approach where each example (x, Y) is decomposed into $|\mathcal{Y}|$ examples, $(x, c, Y(c))$, for all $c \in \mathcal{Y}$, where

$$Y(c) = \begin{cases} 1, & \text{if } c \in Y, \\ -1, & \text{otherwise.} \end{cases}$$

Esuli et al. (2006) proposed an improved version of AdaBoost.MH, called AdaBoost.MH with multiple pivot terms or MP-Boost. The multiple pivot terms, one for each category, are selected at each boosting iteration to give the best possible decision stumps. The final result is a set of classifier committees, one for each category. The method was shown to be a good alternative to AdaBoost.MH at the price of some decrease in classification efficiency. In other words, MP-Boost is more accurate than AdaBoost.MH, but on the computational complexity issue, MP-Boost incurs higher cost of storing the final hypothesis than AdaBoost.MH, and it is slower than AdaBoost.MH on both training and testing.

Clare and King (2001) adapted the C4.5 algorithm for multi-label data by modifying the entropy formula and allowing multiple labels in the leaves of the tree. Gao et al. (2004) extended their binary maximal figure-of-merit learning algorithm to multi-label problems. The method trains all classifiers simultaneously and optimizes performance metrics such as precision and recall. Nonetheless, their discriminant function for classification is still based on individual categories.

Godbole and Sarawagi (2004) presented a technique for combining text features and information about relationships between categories that can be used with any discriminative algorithm. The method has the characteristic of both problem transformation and algorithm adaptation approaches. It stacks two levels of SVM's with heterogeneous features. Each lower level SVM is a single-label, one-against-rest classifier with the original text features as the input. Combining the original text features, the outputs of the lower level SVM's are used as the input of the higher level SVM's which determine the final category label for each document. Zhu et al. (2005) proposed a maximum entropy method in which the correlations among category labels are explicitly considered in the model. Their experiments showed that the method outperforms the combination of single label approach and the technique by Godbole and Sarawagi. Some weaknesses of the method are the computation difficulty and the possibility that the assumption of estimate errors being mutually independent may not be satisfied.

Zhang and Zhou (2005) adapted the k-NN lazy learning algorithm for multi-label data, called ML-kNN. In essence, ML-kNN is a problem transformation method applying the usual k-NN algorithm to learn $|\mathcal{Y}|$ binary classification problems independently, one for each different label c in $\mathcal{Y}$. When consider a label c, examples with label c are positive and the rest are negative. The algorithm finds the k nearest examples of the test example and the label sets of neighboring examples, and then determines whether the test example should be labeled as c or not. ML-kNN is considered an algorithm adaptation because it utilizes prior probabilities and employs the maximum a posteriori (MAP) principle to predict the set of labels of the test example. Additionally, it has the capability of producing a ranking of labels as an output.

Finally, having compiled the results of many research work, Feldman and Sanger (2007) provided general conclusions about the performance of these methods. SVM, AdaBoost, and k-NN were the top performers, but there was insufficient evidence to determine which was the best. Naive-Bayes was the worst among all. For neural network and decision tree classifiers, the results were mixed; they performed poorly in some experiments, and nearly as well as SVM in other experiments.

3.2 Boosting Algorithms

The idea of boosting is to improve the quality of categorization by combining decisions on labels from many classifiers, each obtained by sequentially running a learning algorithm on a different set of examples. Intuitively, if the classifier wrongly classifies an example, the example should be used more often so that the classifier can learn to correctly classify it. Hence, to focus on hard to learn examples, examples in the set are reweighted with greater weight given to those that were misclassified by the previous classifiers.

The first boosting algorithm was presented by Schapire (1990). More efficient variations followed, including Adaptive Boosting, *AdaBoost* (Freund and Schapire 1996; Freund and Schapire 1997), that removed the original algorithm's reliance on unrealistically large training sets. Schapire and Singer (1999) then introduced two extensions, *AdaBoost.MH* and *AdaBoost.MR*. The former induces a classifier that minimizes the Hamming distance between the example's correct class labels and those proposed by the classifier; the latter provides label ranking, seeking to output higher rankings for correct class labels. This lays the foundations for *BoosTexter* that includes four earlier versions of *AdaBoost.MH* and *AdaBoost.MR*.

Several papers have reported successful application of boosting algorithms to text categorization. Schapire and Singer (2000) experimented with Reuters-21578 collection that contains documents from the 1987 Reuters newswire. Diao et al. (2002) applied boosting to decision trees to categorize Yahoo Chinese news. In their approach, they enhanced the efficiency of boosting by Bayesian learning used to select the best splitting point for a decision tree. Cai and Hofmann (2003) showed consistent improvements in categorizing text documents when using AdaBoost to optimally combine weak hypotheses based on term-based and concept-based information. And Chen et al. (2004) sought to categorize Chinese documents from the TCM-MED data set provided by the China Academy of Traditional Chinese Medicine.

Boosting can be used to combine either decision stumps or decision trees. Decision stumps are the simplest case of decision trees which consist of a single decision node and two or more prediction leaves. Boosting decision stumps creates a set of weak hypotheses to be combined. *AdaBoost.MH* is an example of algorithm that boosts decision stumps. On the other hand, boosting decision trees could result in a final combined classifier with a huge number of nodes, which is difficult to interpret. Drucker and Cortes (1996) and Quinlan (1996a) used a decision tree induction as the base learner for boosting and observed that error significantly decreased and the generalization error did not degrade as more classifiers were combined. Freund and Mason (1999) described a different type of classification rule called the Alternating Decision Tree, *ADTree*, which is robust and relatively easy to interpret. Fern and Brodley (2003) presented a relevance-based boosting-style algorithm that builds a lazy decision tree ensemble customized for each test instance. They also introduced a distance-based pruning strategy that improves the accuracy and comprehensibility of both single lazy decision trees and boosted ensembles.

Since boosting is a provably effective method to handle multi-label problems, it is considered as a candidate for the baseline algorithm in the proposed classifier induction system. A derivative of AdaBoost algorithm, *AdaBoost.MH* and *ADTree* are selected for the experiments. We cover briefly here the two algorithms. For a complete discussion on *AdaBoost.MH*, see (Schapire and Singer 1999).

3.2.1 AdaBoost.MH algorithm

AdaBoost is a meta-algorithm that improves or boosts an existing weak classifier. A classifier is viewed as weak if it performs a little better than making random guesses, or, in other words, the classifier makes close to 50% error. Given a weak classifier, AdaBoost improves the performance of the classifier so that there are fewer classification errors. The algorithm progresses iteratively and in each iteration the classifier is improved.

Let $\mathcal{X}$ be a set of examples and $\mathcal{Y}$ be a finite set of class labels. Each $x \in \mathcal{X}$ is assigned a set of class labels, $Y \subseteq \mathcal{Y}$. *AdaBoost.MH* outputs a ranking function, $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, obtained by combining weak hypotheses from many boosting rounds. The boosting algorithm maintains a set of example weights as a distribution D_t over both training examples and labels. This distribution is initially uniform but is updated on each step t, and then passed to the learner that uses it to select a training subset from which to induce $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Each h_t is a "weak hypothesis" that takes the form

$$h(x,c) = \begin{cases} b_{0c}, & \text{if } w \leq a \\ b_{1c}, & \text{if } w > a \end{cases}$$

where b_{0c} and b_{1c} are real numbers. At each boosting iteration, the "weak learner" selects a feature w with the threshold a and values of b_{0c} and b_{1c} in a way that minimizes the Hamming loss. Figure 3.1 gives the details of the algorithm. The end product of this algorithm is a set of C binary classifiers $f(x,c)$, $c = 1, \ldots, C$, that are used to decide on the labels. If the sign of $f(x,c)$, which is a linear combination of the scores from all the weak hypotheses $h_t(x,c)$, is positive, then class label c is assigned to x; otherwise, c is not assigned to x.

Given: $(x_1, Y_1), \ldots, (x_n, Y_n)$ where $x_i \in \mathcal{X}$, $Y_i \subseteq \mathcal{Y}$
Initialize $D_1(i, c) = 1/(nC)$, where C is the size of $\mathcal{Y}$.
Do for $t = 1, \ldots, T$, where T is the number of boosting iterations:

- Pass distribution D_t to weak learner.
- Get weak hypothesis $h_t : \mathcal{X}$ x $\mathcal{Y} \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:
$$D_{t+1}(i, c) = \frac{D_t(i, c) \exp(-\alpha_t Y_i(c) h_t(x_i, c))}{Z_t}$$
where Z_t is a normalization factor (chosen such that D_{t+1} will be a distribution). The parameter α_t is typically chosen as a positive value such that the example-label pairs which are misclassified by h_t receive more weights.

Output the final hypothesis as a weighted vote of the weak hypotheses:

$$f(x, c) = \sum_{t=1}^{T} \alpha_t h_t(x, c).$$

Figure 3.1: AdaBoost.MH algorithm

3.2.2 ADTree algorithm

The Alternating Decision Tree, *ADTree*, is a combination of decision trees with boosting model suggested by Freund and Mason (1999) for the binary classification problems. It generates a different representation of classification rules that can be cast as a weighted majority vote over simple decision trees. The resulting decision trees are often small and concise making it easier to visualize correlations. The *ADTree* structure consists of two components: decision nodes and prediction nodes. Decision nodes specify a predicate condition, and prediction nodes is associated with a real valued number. Each decision node can be seen as a conjunction between a precondition that leads to the decision node and the condition specified in that decision node. The general idea of the *ADTree* algorithm is to grow a tree iteratively, where each decision node is selected by the algorithm based on how well it discriminates between positive and negative examples. Once a decision node is created, the prediction node is determined from how good the decision node discriminates. To classify, *ADTree* maps from each instance to a real valued prediction which is the sum of the prediction scores according to all the simple base rules from preconditions and base conditions. The classification of the instance is based on the sign of the prediction score.

To understand the interaction of decision and prediction nodes, refer to an example in Figure 3.2 which is reproduced from Freund and Mason (1999). The problem domain is heart disease diagnostics for which the goal is to predict whether an individual is healthy (positive) or sick (negative). This alternating tree consists of six decision nodes (rectangles) and 13 prediction nodes (ovals). The numbers within the prediction nodes define contributions to the prediction score. Positive contributions are evidence of a healthy heart, whereas negative contributions are evidence of a heart problem. The root of the tree is associated with the fixed or unconditional contribution of 0.062 meaning that there are slightly more healthy people than sick.

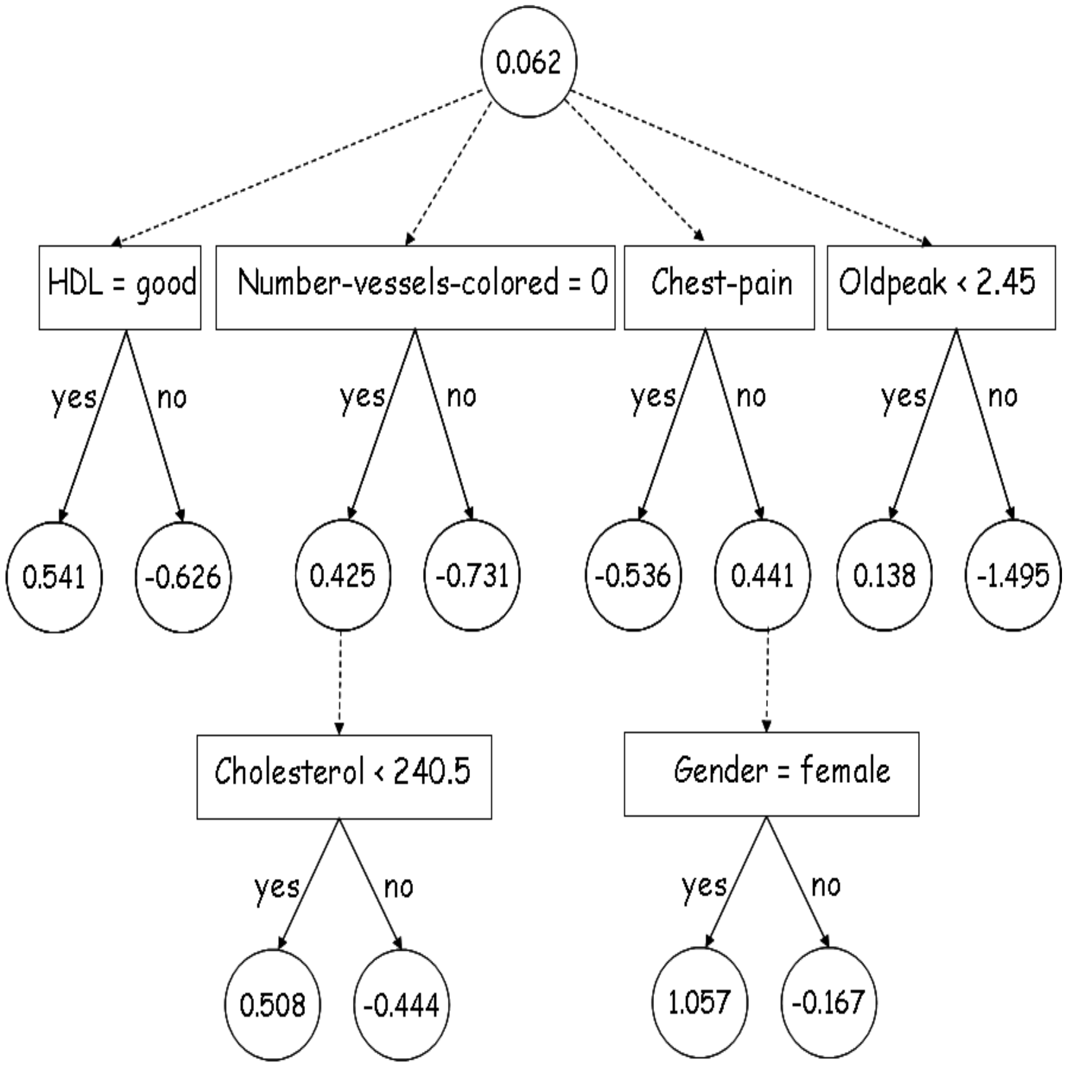


Figure 3.2: An example of ADTree. (Source: Freund and Mason (1999))

This implies that without knowledge of any feature values, an individual should be predicted as healthy, but with low confidence.

To predict the heart health of a particular individual, the conditions given in the tree are tested serially and evidence for or against the health of the person is accumulated as we proceed. More explicitly, starting at the root node, follow all of the dotted arrows that emanate from prediction nodes, but follow only one of the solid-line arrows emanating from a decision node that corresponds to the answer (yes or no) of the individual to the condition at that node. The values in all prediction nodes along the multi-path are summed in order to give the final prediction score whose sign indicates if the person is healthy or not. For example, an individual has HDL = good, number of vessels colored = 0, chest pain = yes, oldpeak = 3, cholesterol = 280, and gender = M. The contributions of relevant prediction nodes are 0.062, 0.541, 0.425, -0.536, -1.495, -0.444 which sum to -1.447. The score -1.447 gives a very confident diagnosis that this individual has a heart problem.

The original formulation of the *ADTree* restricted attention to binary classification problems. Algorithms for decision-tree induction in the multi-class setting were presented by Holmes et al. (2002), and extensions to the multi-label case with heterogeneous input data (discrete and continuous values as well as text data) were proposed by De Comité et al. (2001, 2003).

3.3 C4.5 Algorithm

Decision trees are one of the most popular methods for classification because they provide simple and interpretable rules. A decision tree learning creates from data a predictive model having a tree structure; nodes in the tree represent features, with branches representing conjunctions of features that lead to leaves or terminal nodes representing classifications. Thus, a decision tree can be viewed as a partitioning of the instance space into smaller segments such that each partition, represented by a leaf, contains the instances that are homogeneous and are expected to belong to the same class. Determining the class of an instance from the decision tree is then a matter of tracing the path of nodes and branches starting at the root node of the tree to the terminating leaf.

C4.5 is a decision tree generating algorithm developed by Quinlan (1993) and modified further later by Quinlan (1996b). It is an extension of Quinlan's earlier

ID3 induction algorithm (Quinlan et al. 1986) with a number of improvements to account for unknown attribute values, attributes with differing costs, and bias towards continuous attributes with numerous distinct values. In addition, it has several new and interesting features including a new criterion for determining the best partitioning of the examples at each decision tree node, "pruned" decision trees to avoid overfitting the data, and the ability to derive classification rules from the unpruned decision tree.

C4.5 builds decision trees from training data using the information entropy concept to measure how informative a node is. It examines the information gain, which is the effective decrease in entropy, that results from choosing a feature to split the data at a particular node in the tree. The feature having the highest information gain is the best for discriminating among cases at that node, so it will be chosen to make the decision.

C4.5 is a good choice for practical classification due to the ease of its interpretability as well as its ability to deal with numeric attributes, missing values, and noisy data. Quinlan (1996b) showed that the new version of *C4.5* gave smaller decision trees with higher predictive accuracies. In their IP traffic flow classification study, Williams et al. (2006) compared four different machine learning algorithms, Bayesian network, naive Bayes, naive Bayes tree, and *C4.5* decision tree, and concluded that *C4.5* was the best suited for real-time classification tasks. All algorithms provided very similar classification accuracy, however, the *C4.5* algorithm was significantly faster than other methods in terms of classification speed.

3.4 k-Nearest Neighbor Algorithm

Behind the idea of k-NN lies an assumption that the class of an object is similar to the class of other objects that are nearby in the feature space. To classify a test example, the algorithm finds the k nearest or most similar neighbors of this new example from a given set of training examples. Similarity is measured by the Euclidean distance or the cosine similarity measure between two example vectors.

The Euclidean distance between 2 examples $x_1 = (x_{11}, x_{12}, \ldots, x_{1p})$ and $x_2 = (x_{21}, x_{22}, \ldots, x_{2p})$ is

$$dist(x_1, x_2) = \sqrt{\sum_{j=1}^{p} (x_{1j} - x_{2j})^2}$$

and the cosine similarity between x_1 and x_2 is

$$sim(x_1, x_2) = \frac{\sum_{j=1}^{p} x_{1j} x_{2j}}{\sqrt{\sum_{j=1}^{p} x_{1j}^2}\sqrt{\sum_{j=1}^{p} x_{2j}^2}}$$

The traditional k-NN method deals with single label classification problems. Normally the classification uses majority vote among the classification of these neighbors, i.e., the most frequent class label among the k closest training examples is assigned to the new instance. Some researchers prefer predicting the category of an instance using other alternative criteria such as the class that has maximal sum of similarity (Li et al. 2003) or the class that has highest confidence score (Joachims 1998). k-NN is often a good choice for classification task when simplicity and accuracy are the predominant issues. The algorithm is easy to implement, but it is very computationally intensive and requires training examples to always be retained. Many optimizations for k-NN have been proposed over the years (see, e.g., Zhang and Srihari (2004) and Ramasubramanian and Paliwal (2000)); these methods generally seek to reduce the number of distances actually computed.

The simple k-NN method needs modification so that it can assign multiple categories to any document. Some variants of k-NN to handle multi-label problems have been derived. Yang and Liu (1999) and Lewis et al. (2004) studied the weighted k-NN method where the similarity score of each neighbor document is used as the weight of the categories of the neighbor document. For each category, the sum of all the weights from k neighbors is taken as the likelihood score of that category with respect to the test document. The scores of candidate categories can be sorted to provide a ranked list of categories. Alternatively, thresholding on these scores will give decision rules that produce binary category assignments.

CHAPTER 4

Classifier Induction for a Large Feature Set

Our research interest is in the induction of multi-label classifiers in the situation where the information set is exceptionally large with a very high dimensional feature space and a large number of classes. Inauspiciously the application of many machine learning methods could be restricted on a large-scale database because of limited computational resources, such as CPU time, memory, and storage. The processing of data in the whole data set may be impractical or not feasible owing to computational and time complexities, even with the use of highly efficient learning algorithms. This circumstance leads us to explore an alternative solution to the handling and modeling of a huge amount of data.

This chapter begins with the introduction to the EUROVOC thesaurus, a large-scale database used in testing the proposed solution. The size of this database was a real challenge we confronted when trying to apply some existing learning algorithms to the data. An ensemble method to handle the problem is explored, which leads to the proposed classifier induction system. The system is the fusion of multiple multi-label classifiers consisting of 2 steps, inducing subclassifiers and combining subclassifiers. We describe how a baseline algorithm for subclassifier induction is selected. The discussion of the subclassifier combination methods is postponed till the next chapter.

4.1 EUROVOC Data

One example of a massive real-world document collection is the EUROVOC classification data. EUROVOC [1] is a multilingual, polythematic thesaurus developed in the course of close cooperation between the European Parliament, the European Commissions Publications Office, and the national organizations of the European Union (EU) member states. The EUROVOC thesaurus focuses on many different fields of interest to EU such as law and legislation, economics, trade, education and communications, science, employment, transport, environment, agriculture, forestry and fisheries, foodstuffs, production, technology and research, energy, international organizations, industry, employment and working conditions, business and competition, and many others. The thesaurus provides indexing of the documents available in the documentation systems of the European institutions, and is used in libraries and document centers of national parliaments as well as other governmental and private organizations of member and non-member countries of the EU.

We were given access to a part of EUROVOC classification system consisting of 78,599 documents with over 5,000 descriptor terms (class labels) organized hierarchically into eight levels, the top-most level having 30 different classes. Among 78,599 documents, 10,868 documents are not assigned any descriptors. Excluding those unlabeled documents leaves us with 67,731 documents in 5,452 fields or classes. Each document can belong to more than one field, with some documents belonging to as many as 30 fields. Each document is described by 105,355 features representing the frequency of prespecified words in the document. The file size of unprocessed data where all data with value 0 are omitted is about 3GB. After filling necessary data values, the total size of data files becomes more than 16GB.

[1] `http://europa.eu/eurovoc/` and `http://langtech.jrc.it/Eurovoc.html`

For information retrieval purpose, it is desirable to have decision rules that enable the classification of a document into areas or categories where it belongs to. But because of a huge number of classes and features that EUROVOC contains, it is exceedingly difficult to do text categorization of these data using currently available methods. Computer time and resource requirements to process this vast amount of data would hamper the task of finding good classifiers. For instance, it took 38 hours to train a small subset of EUROVOC classification data that has 19,095 documents, 4,004 classes, and 100 features using *ADTree* learning algorithm with 10 rounds of boosting on an AMD 64 bits Dual Core X2 3800+, 2GB memory. Thus for more than 100,000 features that the whole EUROVOC database has, the situation is far worse.

Upon proposing the new classifier induction technique for the problem with a large feature set, the primary concern is whether the technique has any computational advantage over traditional methods. Unfortunately, the large data volume of this database virtually excludes the possibility of performing comparison studies on a personal computer using the entire database. The fact that each experimental run takes many days makes it impossible to go through the hundreds of experiments needed for statistically justified conclusions. To this end the decision is to experiment with a small portion of the entire EUROVOC database. This simplified version of the EUROVOC database contains 10,000 documents described by a set of 4,000 features and classified into 30 major classes at the top-level of the classification hierarchy. The features were extracted on the basis of Document Frequency criterion, which is a simple unsupervised feature selection method found to be effective for text categorization and have low computational cost (Yang and Pedersen 1997; Calvo et al. 2004). Document frequency is the number of documents in which a term occurs in a data set, and for our study, the 4,000 features were randomly selected from those having document frequency larger than 50.

The data are summarized in Figure 4.1 that gives the number of documents in each of the 30 classes. One can see that this database is very highly imbalanced. Figure 4.2 shows how many documents belong to one class, how many of them belong to two classes, and so on—the highest number of class labels for a single document is, in this sample, 15. Near 90% of the documents have 2 to 5 labels and only a few, 2.38%, are single-labeled. The label cardinality of this experimental data set is 3.6, and the label density is 0.12. In other words, in average each document has 3.6 labels, or about 12% of the total number of labels.

4.2 A Solution - Fusion of Multiple Classifiers

The amount of data was a pressing issue when working on EUROVOC data set. The early attempts to train a small fraction of EUROVOC data revealed a potential problem about the tremendous time it took to induce a classifier. Some learning methods even failed to complete their jobs after running continuously for a few days on a PC. At last we were able to gather useful information from the experiments with *AdaBoost.MH* algorithm, which is a subsystem of *BoosTexter* [2], a program written by Allwein, Schapire, and Singer. The experiments were carried out for feature sets of different sizes, 100, 200, 500, 1,000, 2,000, and 4,000 features. There were 10,000 documents and 30 classes. The number of boosting rounds was set to 10% of the number of features used (i.e., 10, 20, 50, 100, 200, and 400 boosting rounds for 100, 200, 500, 1,000, 2,000, and 4,000 features, respectively). 5-fold cross validation (CV) was used to assess the performance of the method, although its accuracy will not be discussed at this time.

[2]available for free for non-commercial research or educational purposes at `http://www.cs.princeton.edu/~schapire/boostexter.html`

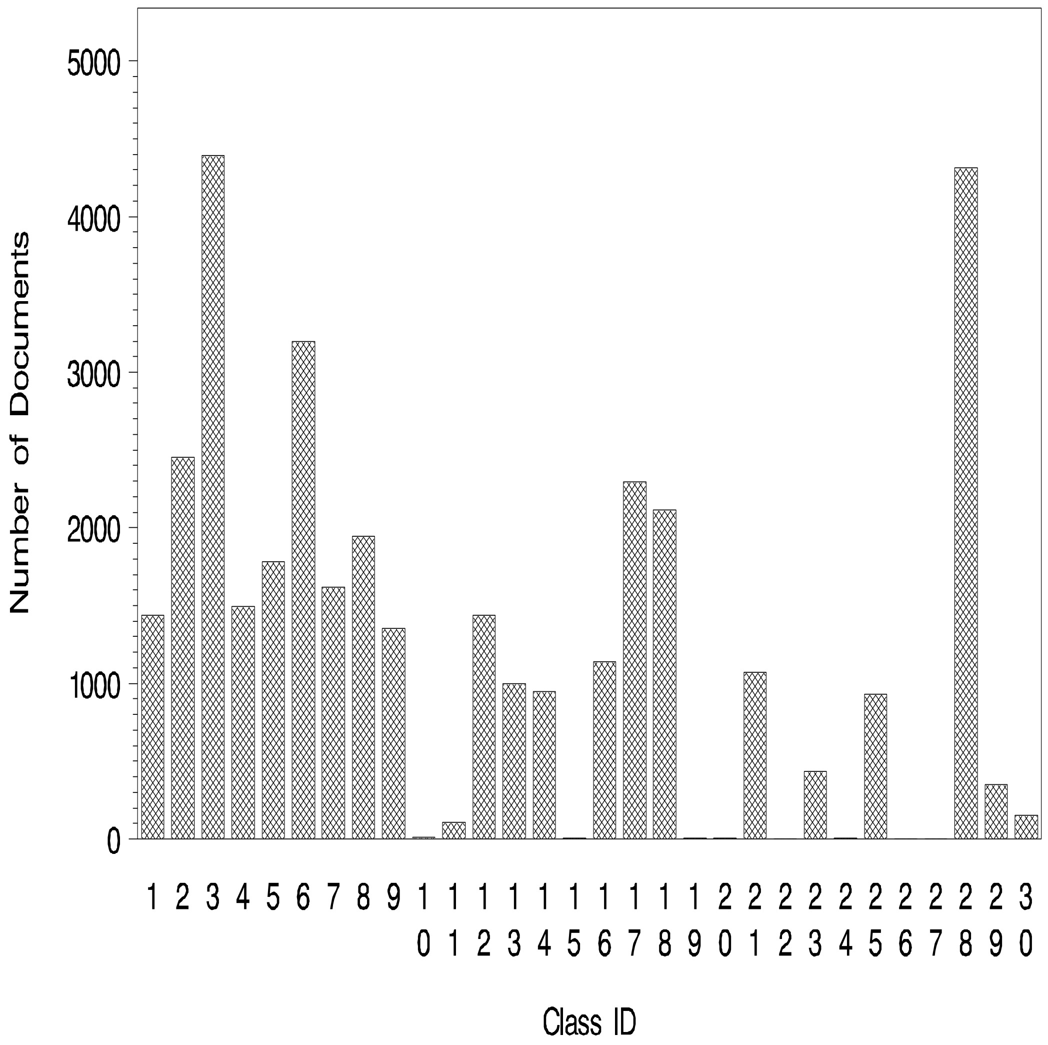
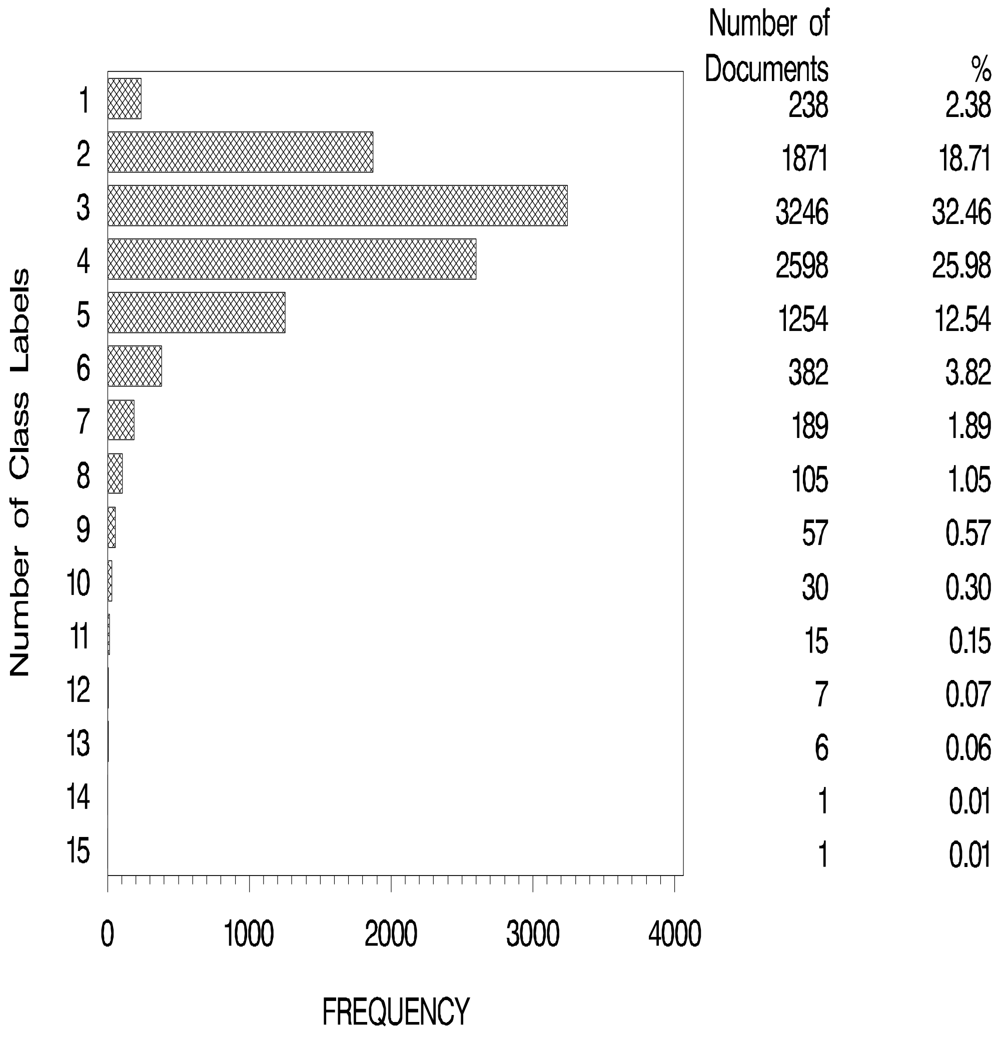


Figure 4.1: Class distribution of 10,000 documents in experimental data, non-disjoint classes.



Number of Class Labels	Number of Documents	%
1	238	2.38
2	1871	18.71
3	3246	32.46
4	2598	25.98
5	1254	12.54
6	382	3.82
7	189	1.89
8	105	1.05
9	57	0.57
10	30	0.30
11	15	0.15
12	7	0.07
13	6	0.06
14	1	0.01
15	1	0.01

Figure 4.2: Number of documents in the experimental data having different number of class labels.

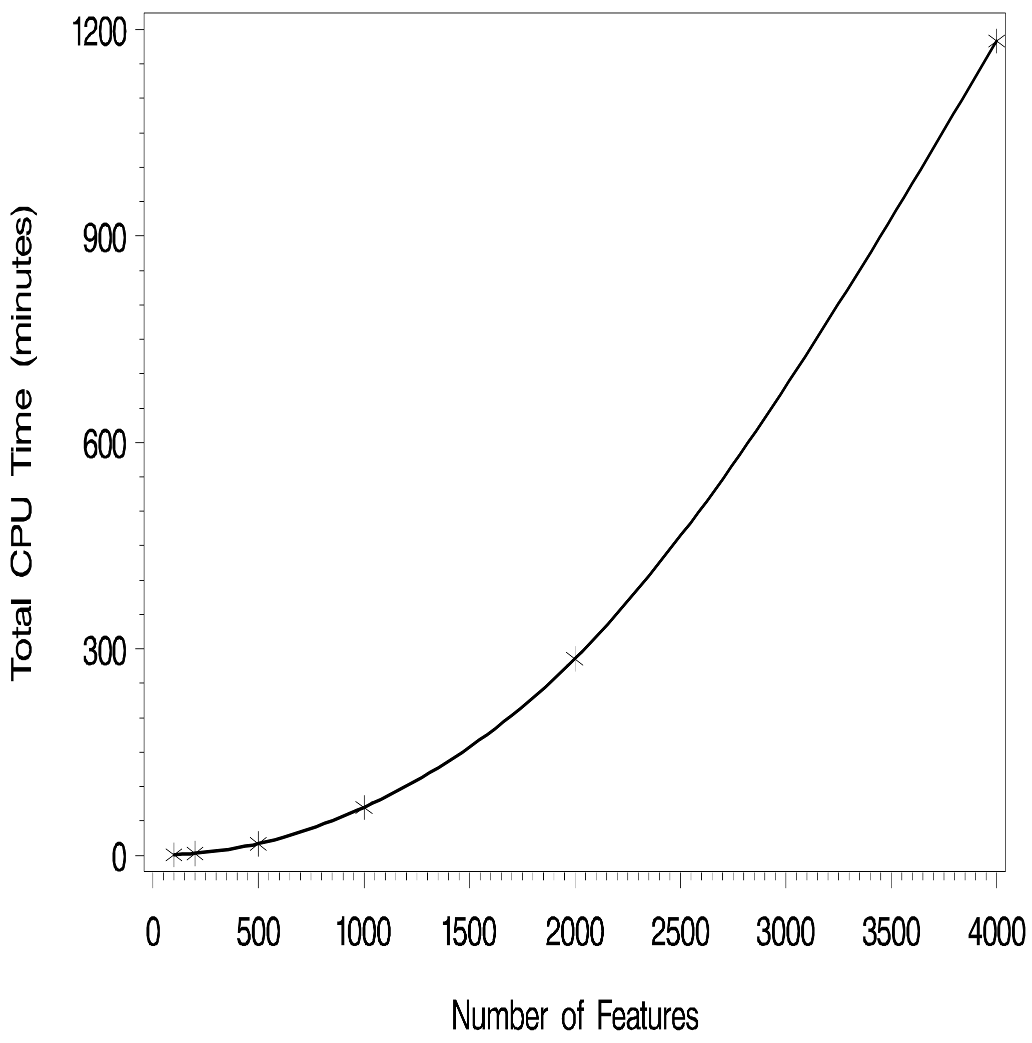


Figure 4.3: The CPU time in minutes of AdaBoost.MH for varying number of features, 5-fold CV.

Figure 4.3 shows the total CPU time required by *AdaBoost.MH* to learn from the experimental data set. It is clearly seen that the run time for induction increases very fast, faster than the linear growth rate, with the increasing number of features. From the graph one may project that it will take many days to induce a classifier if one is to use, for example, 10,000 features which amounts to only 10% of the total number of features in the EUROVOC data, and this is quite discouraging.

However, this preliminary study points us to a possible alternative solution to handling a large feature set. One can observe in Figure 4.1 that the induction time for 1,000 features is more than twice the induction time for 500 features. Similarly, the induction time for 4,000 features is also more than twice the time for 2,000 features, and much more than 4 times the time for 1,000 features, and so on. That is, in general, if a large feature set is partition into small feature subsets, the total time for induction of all small feature subsets is far less than the induction time of one large feature set. To be more concrete, suppose a feature set $\mathcal{X}$ is partitioned into N disjoint feature subsets, $\mathcal{X}_1$,$\mathcal{X}_2, \ldots, \mathcal{X}_\mathcal{N}$, $\bigcup_{j=1}^{N} \mathcal{X}_j = \mathcal{X}$. Let T_j be the time needed to induce a classifier from the j-th feature subset $\mathcal{X}_j$ and T is the time needed to induce a classifier from all features $\mathcal{X}$. Then the total time to induce N classifiers, one from each of $\mathcal{X}_1$, $\mathcal{X}_2, \ldots, \mathcal{X}_\mathcal{N}$ is $\sum_{j=1}^{N} T_j$, and $\sum_{j=1}^{N} T_j \ll T$. Even in the case that some feature subsets are overlapping, i.e., $\mathcal{X}_j \cap \mathcal{X}_k \neq \emptyset$ for some $j \neq k$, the total induction time of classifiers from many small feature subsets is still much less than the induction time needed when all features are used at once. This will be seen from the experiments in the following chapters.

Therefore, within the current context, a feasible and efficient procedure for rule induction on a large data set would be to divide the large data set into smaller sets so that each small portion of data can be managed more easily. First, features are separated into subsets and then the large data set is split accordingly. Some good learning algorithm can now be applied to each feature subset. Later, after all small data sets are processed, their results will be combined by some data fusion technique to obtain the final classification result. This strategy will not require a high performance computer, e.g., memory requirements would be less; indeed, the smaller data sets can even be processed concurrently on multiple computers to finish the whole classification task faster.

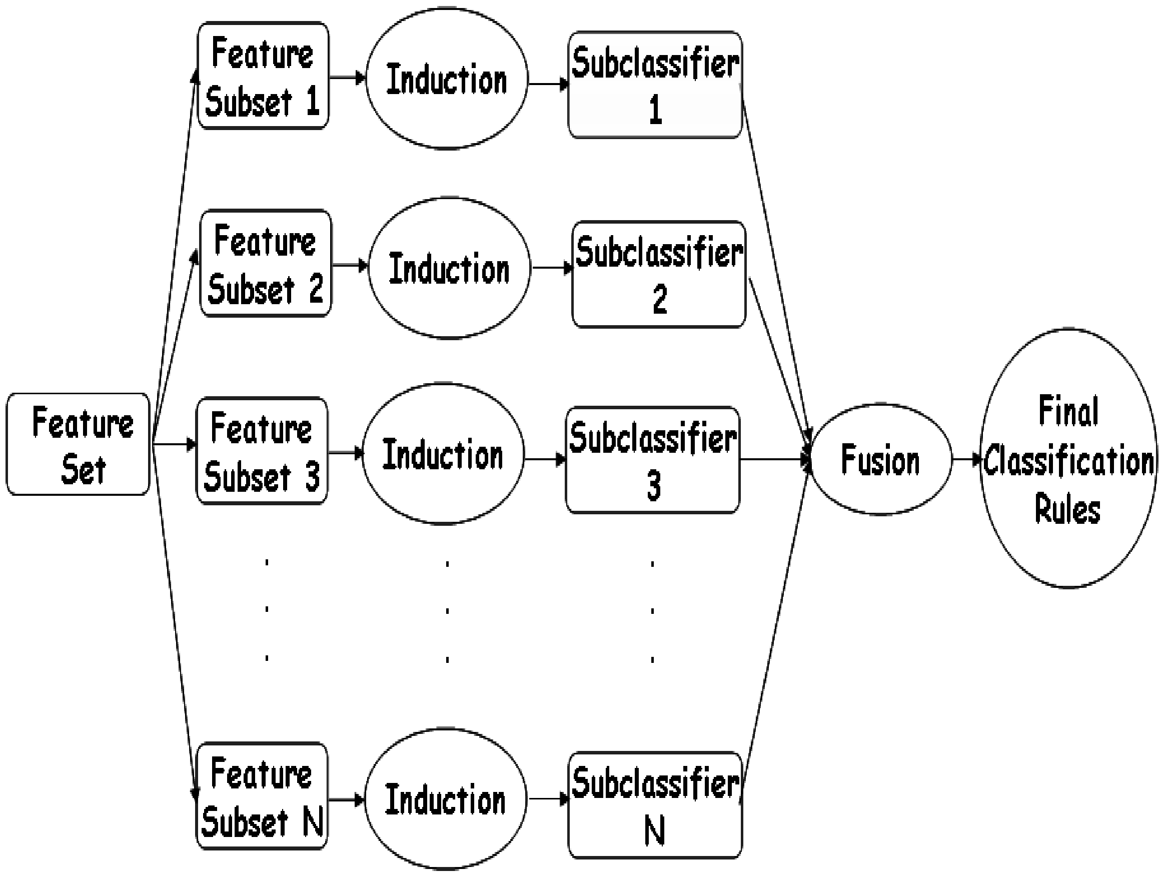


Figure 4.4: Schematic of induction process for training data with large feature set

The conceptual framework of the proposed method for multi-label classifier induction is shown in Figure 4.4. The system is composed of two components, learning method to induce subclassifiers from different feature subsets and fusion algorithm to combine subclassifiers' decisions. Specifically, in this approach, a multi-label learning method is selected as a baseline algorithm for rule induction. Once the original feature set is decomposed into N feature subsets, run the "baseline induction algorithm" (BIA) independently on different feature subsets to obtain several subclassifiers, one for each feature subset. After that, combine outputs of these subclassifiers by exploiting a "master algorithm" that creates the final set of class labels to be returned to the user.

4.3 Choosing a Baseline Induction Algorithm

An important element of the proposed mechanism is the baseline algorithm used to induce multi-label subclassifiers. Every subclassifier has to decide on the plausibility of each label such that decisions of all subclassifiers are fused by a master algorithm afterward. In principle, any learning algorithm that is able to produce ranked labels result can be used as a baseline learner. However, for practical applications, it is natural to choose an induction algorithm that is both fast and accurate in order to ensure good final classification result. Therefore, during this initial stage, some experiments were conducted in search of a good choice for the baseline algorithm.

There are two existing programs available from the web that can induce from data the information needed for fusion: *ADTree* [3] and *BoosTexter*. For *BoosTexter* program, we worked specifically with *AdaBoost.MH* subsystem. The selection between these two candidates were based on two criteria: classification performance and computational costs. *AdaBoost.MH* beats *ADTree* on both criteria, and thus is a good choice as BIA. To develop a thorough understanding of the behavior of *AdaBoost.MH*, we explored *AdaBoost.MH* further by comparing it to two other learning methods, k-NN and *C4.5*. These two methods were modified so that they can work on multi-labeled examples. All the experiments and results are detailed in the following sections.

4.3.1 Two multi-label algorithms: AdaBoost.MH vs. ADTree

It was expected that the baseline algorithm would have to operate with feature subsets consisting at least of 100 features, but probably not more than 500 features. Therefore, we ran a simple experiment for the following numbers of features: 100,

[3] written by De Comité and Devigne and available from `http://www.grappa.univ-lille3.fr/grappa/en_index.php3?info=software`

200, and 500. The number of boosting rounds for both *AdaBoost.MH* and *ADTree* was set to 10% of the number of features used. For the sake of statistical credibility, all results were averaged over 5-fold cross validation—the training set then always contained 8,000 documents and the testing set 2,000 documents. Table 4.1 summarizes the results for four multi-label performance criteria from Section 2.3, giving the average values and the standard deviations obtained from 5-fold CV. The performance of both algorithms improves slightly with more features. Boldfaced font highlights the cases where one of the candidates is significantly better than the other when they are subject to paired t-tests. One can see that *AdaBoost.MH* has systematically outperformed *ADTree* on three out of the four criteria, albeit only by a narrow margin.

Table 4.1: Mean/Standard deviation (5-fold CV) of 4 evaluation measures for AdaBoost.MH and ADTree. Bold items indicate that AdaBoost.MH is significantly better at 0.05 level over ADTree.

# of Features	Average Precision	Coverage	Hamming Loss	One Error
100				
AdaBoost.MH	**0.523**	**10.7**	**0.116**	0.47
	±0.005	±0.1	±0.001	±0.01
ADTree	0.510	10.9	0.118	0.48
	±0.006	±0.2	±0.001	±0.01
200				
AdaBoost.MH	**0.560**	**10.2**	**0.113**	0.41
	±0.007	±0.1	±0.001	±0.01
ADTree	0.540	10.5	0.115	0.43
	±0.007	±0.1	±0.001	±0.02
500				
AdaBoost.MH	**0.589**	**9.6**	**0.109**	**0.39**
	±0.007	±0.1	±0.001	±0.02
ADTree	0.565	9.9	0.112	0.42
	±0.006	±0.1	±0.001	±0.01

The difference in performance of the two algorithms being small, we further explored the computational costs. Here, the situation was different. Table 4.2 compares the CPU time consumed by these two programs when run on data described by different numbers of features. The results obtained were the total CPU time from 5-fold CV. The induction time of *ADTree* also rose very quickly as the number of features increased, similar to what has been observed from *AdaBoost.MH* in Figure 4.1, but *ADTree* spent much more time compared to *AdaBoost.MH* for the same number of features. Since the time needed by *ADTree* was one to two orders of magnitude higher than the time needed by *AdaBoost.MH*, statistical evaluation would be superfluous. Hence, on the basis of its performance and processing time requirement, we decided to choose *AdaBoost.MH* as the baseline induction algorithm.

Table 4.2: Total CPU time (5-fold CV) in minutes required by AdaBoost.MH and ADTree.

# of Features	AdaBoost.MH	ADTree
100	1	15
200	3	116
500	16	1,682

4.3.2 Comparing AdaBoost.MH to k-NN and C4.5

In this section, the experiments were done with three programs on the whole simplified database with all 4,000 features: (1) the code of *AdaBoost.MH*, fixing its number of boosting rounds at 400; (2) Weka's reimplementation of *C4.5* [4] (Witten and Frank 2005); and (3) our own implementation of the k-NN classifier for multi-label problems. Again, the three candidates were evaluated in terms of computational costs and classification performance.

[4]Weka is a collection of machine learning algorithms for data mining tasks, available for download at http://www.cs.waikato.ac.nz/ml/weka/

C4.5 is an algorithm that induces classification rules in the form of decision trees for multi-class, single-label classification problems. In order to run *C4.5* under the multi-label setting, we take the problem transformation approach to decompose the multi-label problem into multiple, independent binary classification problems. That is, for the problem having C predefined classes, C binary classifiers are independently built, one for each class. These separate binary classifiers are then used to predict whether a document belongs to the corresponding class. The final labels assigned to the document are obtained by aggregating the label assignments from all the binary classifiers.

k-NN is amongst the simplest of all machine learning algorithms for single-label classification problems. We modify the traditional k-NN algorithm for multi-label problems in the following way. To identify the labels of an example x, first find in the given training data set the k nearest neighbors of x. Let I be the set of indices of these k nearest neighbors. Assuming all training examples were correctly labeled, an easy and straightforward approach to assigning labels to the example x is to give all labels appearing in at least one of the k nearest neighbors. That is,

$$\text{“Assign label } c \text{ to } x \quad \text{if } \; c \in \bigcup_{i \in I} Y_i\text{”}.$$

where Y_i is the set of class labels of an example i in the training data set, and $\bigcup_{i \in I} Y_i$ is the union of the label sets of all k nearest neighbor documents. This assignment generally yields high recall rate and low precision particularly when k is large. Nevertheless, this rule is used simply to provide a basis for comparison with other methods.

While *C4.5* and *AdaBoost.MH* were run with default settings, the experiment with k-NN was done with different values of k. The parameter k was varied from 5 to 200. Figures 4.5 and 4.6 show the macro-average and micro-average of three evaluation measures, precision, recall, and F_1, of k-NN method for different k values.

Macro-average recall was around 60% when $k = 5$ and it increased with the k value reaching about 90% when k was 80, or the number of nearest neighbors equaled to 1% of the total number of training examples. Micro-average recall exhibited a similar pattern, but its value was higher and it approached 100% very fast. This is not surprising since almost all of the labels would be assigned to the documents when more and more examples are included in the nearest neighbor set. This also explains the decrease in the precision, both macro-averaging and micro-averaging, when k increased. The precision was quite low, staying below 25% for all k's. The F_1 measure was dominated by the precision; the two curves show almost identical shape. F_1 monotonically decreased because the percentage of decrease in the precision was larger than the percentage of increase in the recall. It is clear that $k = 5$ produced the best F_1 measure. Hamming loss in Figure 4.7 also indicates $k = 5$ performed the best. This measure will reach its maximum value when all the labels are assigned to every document which happens when k is sufficiently large.

In all, k-NN achieved the best results when $k = 5$. A close look at some of the results from $k = 5$ reveals that each of the 5 nearest neighbor documents has 4 to 5 labels. The union of these label sets has about 15 members or half of the total number of labels in the data set. Therefore the recall rate is high even k is only 5. It should be noted that the majority of the documents in this data set have 3 or 4 labels (see Figure 4.2).

Next, look at how the three learning algorithms compare on efficiency. Shown in Table 4.3, the induction time of 8,000 training documents by *AdaBoost.MH* and *C4.5* was roughly 3.5 hours and 25 hours, respectively. So *AdaBoost.MH* ran seven times faster than *C4.5*. The induction costs of the k-NN were negligible since there is no training (just storing the training examples), but it places heavy load on classification time. The classifier needed on average more than 3 hours to classify the 2,000

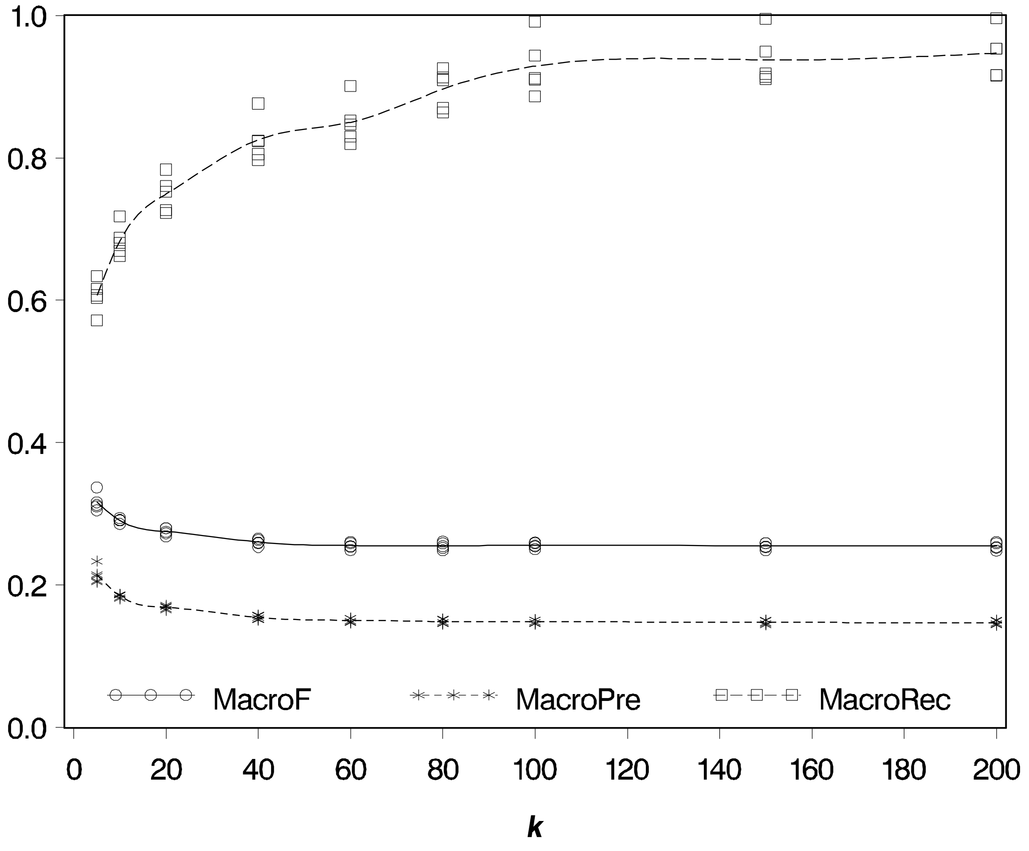


Figure 4.5: Macro-average recall, precision, and F_1 of the k-nearest neighbor algorithm. The smooth curves are the averages of the 5-fold CV.

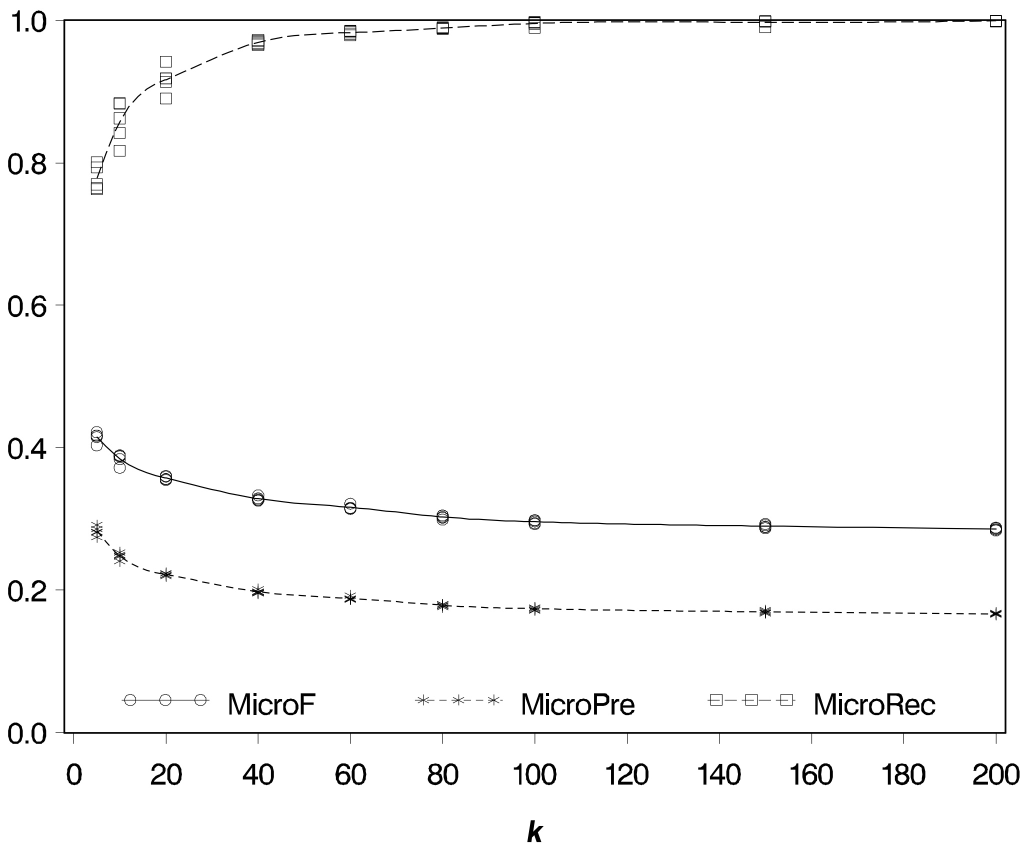


Figure 4.6: Micro-average recall, precision, and F_1 of the k-nearest neighbor algorithm. The smooth curves are the averages of the 5-fold CV.

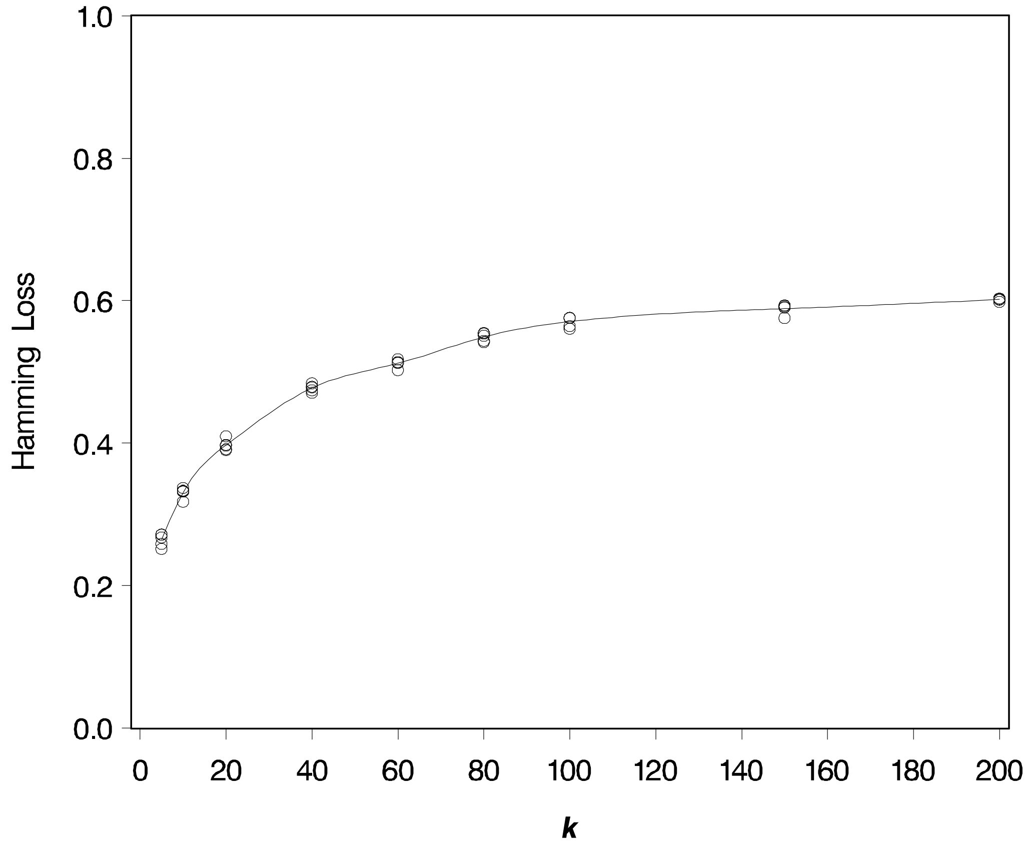


Figure 4.7: Hamming loss of the k-nearest neighbor algorithm.

Table 4.3: Time consumed by three machine learning techniques.

Algorithm	Induction Time (min.)	Classification Time (min.)
AdaBoost.MH	1,105	-
C4.5	7,624	-
5-NN	-	1,005

testing documents when $k = 5$ was used. This is almost prohibitive in a domain like EUROVOC that is ultimately expected to grow to millions of documents.

To study the classification performance of these methods on testing data, the result of k-NN, when $k = 5$, is compared to *AdaBoost.MH* and *C4.5* results. Table 4.4 shows the Hamming losses, macro- and micro-average precisions, recalls, and F_1-metrics. The analysis of variance indicates that the observed differences in the performance among the three methods are statistically significant. The multiple comparisons by the Ryan-Einot-Gabriel-Welsch multiple range test (Hsu 1996) suggest the following conclusions: 1) *AdaBoost.MH* outperformed the other two methods with regard to precision, 2) *AdaBoost.MH* was comparable to *C4.5* based on Hamming loss, recall, and F_1, 3) 5-NN had much higher Hamming loss and poorer precision, but better recall rate compared to *AdaBoost.MH* and *C4.5*, 4) 5-NN had higher values of F_1 than others, and 5) *C4.5* was the worst performer among the three in almost every aspect. Therefore, these empirical results obviously support the use as a BIA of *AdaBoost.MH*.

Table 4.4: Classification performance of three machine learning techniques on independent testing data, estimated by 5-fold CV. (The same number in the circle indicates the methods are not different at 0.05 significance level. The smaller number means the better performance.)

Method	Hamming Loss		
AdaBoost.MH	① 0.1028 ± 0.001		
C4.5	① 0.1060 ± 0.002		
5-NN	② 0.2645 ± 0.009		
	Macro F_1	Macro Precision	Macro Recall
AdaBoost.MH	② 0.2817 ± 0.009	① 0.5675 ± 0.016	② 0.1874 ± 0.007
C4.5	② 0.2829 ± 0.020	② 0.4997 ± 0.030	② 0.1976 ± 0.017
5-NN	① 0.3165 ± 0.012	③ 0.2146 ± 0.012	① 0.6066 ± 0.023
	Micro F_1	Micro Precision	Micro Recall
AdaBoost.MH	② 0.3850 ± 0.005	① 0.6886 ± 0.009	② 0.2672 ± 0.004
C4.5	① ② 0.3959 ± 0.028	② 0.6388 ± 0.009	② 0.2880 ± 0.033
5-NN	① 0.4149 ± 0.007	③ 0.2829 ± 0.006	① 0.7784 ± 0.018

CHAPTER 5

Fusion of Multi-label Subclassifiers

A key step of the classifier induction system for multi-labeled examples presented in the previous chapter is the fusion of outputs from multiple multi-label subclassifiers. In this approach, a master algorithm combines the "testimonies" of an ensemble of subclassifiers, obtained by a BIA, each time using a different subset of the features. *AdaBoost.MH* has been selected as the BIA. What is needed now is a master algorithm to combine subclassifiers' decisions in order to generate a consensus about the class labels. This chapter discusses the development of a master algorithm.

Classifier fusion, also referred to as classifier combination or decision fusion, provides a way to combine information to help in decision making from many sources, or, in our case, multiple subclassifiers. First off, a review of some research work on classifier combination and a number of combination methods that are available are presented. A short introduction to the Dempster-Shafer theory of evidence (DST) is given to provide a background for our proposed subclassifier combination method. Several researchers have used DST to combine classifiers for single-label classification. Here we describe in detail how this theory can be applied to multi-label problems and then create a master algorithm that is built around the Dempster-Shafer combination.

5.1 Review of Classifier Fusion Methods

Combining together multiple classifiers that are individually trained to obtain a single, high-performance, classification system is a topic that has been extensively researched in the machine learning communities for several decades. Fusion has been proven useful when several classification methods are used to construct classifiers from a set of data. The facts are that different methods can give different results with different degrees of success, and no single classifier is perfect. Research has shown that an ensemble of classifiers can produce (though not always) more accurate decisions than what an individual classifier can provide, depending on the quality and the diversity of the ensemble members (Bell et al. 2005; Larkey and Croft 1996). An example of real applications is face recognition by Lu et al. (2003) who used a combination of different face classifiers to integrate the complementary information that leads to more accurate face recognition than that made by any one of the individual classifiers.

Ensembles can be implemented in a variety of different ways. An overview of various classifier fusion techniques was given by Ruta and Gabrys (2000). There are two general groups of classifier fusion methods. The methods associated with the first group generally operate on classifiers and find a single best classifier or a selected group of classifiers. They emphasize on a development of the classifier structure. The second group of methods operate mainly on classifiers' outputs, and effectively calculate the combination of classifiers' outputs.

Roli and Giacinto (2002) discussed the problem of multiple classifier system (MCS) development and experimentally assessed six different design methods based on the *overproduce and choose* design paradigm. The basic idea of *overproduce and choose* is to produce an initial large set of candidate classifier ensembles, and then select the ensemble that can be combined to achieve highest accuracy. This paradigm provides a practical and effective solution, however, an optimal MCS design is not guaranteed.

Kubat and Cooperson (2001) also employed a similar approach, but with a more elaborate selection mechanism.

Apart from the system design issue, the choice of strategy used in combining the results of different classifiers is very important as well. One simple strategy that is frequently used is majority voting. Rahman et al. (2002) reviewed several majority voting systems and their variations and found that these techniques can achieve very robust performance and often compare favorably with many complex and sophisticated systems. Some variations of majority voting systems are (Fürnkranz 2002; Rahman et al. 2002; Jain et al. 2004):

- Simple majority voting. This technique is to give each classifier one vote, and predict the class that receives the most votes. Ties are broken in favor of larger classes, and predictions with 0 confidence score are ignored.

- Weighted majority voting. This voting scheme is identical to the previous technique except that the accuracy of each classifier is used to weight its vote. When asked to make a prediction, the votes from all classifiers are combined based on their associated weights, and the class label with the highest weighted vote is selected. Weighted majority voting provides a simple and effective method when one of the classifiers is known to perform well.

- Weighted sum. This voting scheme uses the accuracy of each classifier to weight confidence scores of possible classes. The weighted sum of confidence scores from all classifiers provides the overall confidence in the label. The label that receives the highest sum of weighted confidence score is predicted. With this scheme, a few predictions with high confidence scores may over-rule a larger number of predictions with lower scores.

- Weighted normalized sum. This voting scheme is identical to the weighted sum, except that the confidence scores are first normalized in a way that distributes a total weight of 1 among the different candidate classes for each example. This is to ensure that the confidence scores associated with each class can be interpreted as class probability estimates.

- Maximum confidence. This method simply chooses the class prediction that receives the highest confidence among all members of the ensemble and predicts that class. This is an attempt to use only the most accurate of all applicable rules to classify an instance.

- Restricted majority voting. This method selects the best decision delivered by the best classifier.

A large group of classifier combination methods work on classifiers that perform soft categorization and produce soft outputs which are real-valued. These soft outputs are usually referred to as measures of evidence and are used to describe information uncertainty; more familiar terms would be probability, possibility, belief, or plausibility. Some fusion methods in this group are Bayesian fusion and Dempster-Shafer fusion (Challa and Koks 2004). Bayesian fusion combines classifiers based on the multiplicative rule on the posterior probabilities of individual classifiers. Langley et al. (1992) showed that Bayesian fusion works well in many real world problems despite the fact that the conditionally independent assumption is violated. As opposed to well-understood probability of Bayes theory, Dempster-Shafer theory deals with measures of belief. It allows us to cast doubt on the state of a system as unknown in addition to true or false as in Bayes theory. Details of Dempster-Shafer fusion will be presented in the next section.

Combining outputs from different classifiers has shown considerable promise in machine learning generally. However, combining several text categorizers has had mixed success (Uren and Addis 2002). Some researchers reported improvements with combined systems, and some reported no improvements. In other words, in text categorization applications, combinations of multiple classifiers do not always improve the classification accuracy compared to the best individual categorizer. Below are some classifier combination methods that are used in text categorization.

Bennett et al. (2005) proposed a probabilistic method for combining classifiers that hinges on learning the context-sensitive reliability of contributing classifiers. The method harnesses reliability indicators which are variables that provide signals about the performance of classifiers in different situations. These variables are used to weave together multiple classifiers in a coherent probabilistic manner to boost or improve overall accuracy. The authors presented procedures for building metaclassifiers that take into consideration both reliability indicators and outputs from base-level classifiers, and reviewed a set of comparative studies undertaken to evaluate the methodology. The empirical evaluations supported the conclusion that a simple majority vote in situations where one of the classifiers performs strongly can weaken the best classifier's performance. In contrast, their methodology was competitive and produced the top performer in most instances.

Fürnkranz (2002) introduced hyperlink ensembles for classifying hypertext documents. Instead of using the text on a target page to derive features for training a classifier, he suggested to use portions of texts from all pages that have a hyperlink pointing to the target page. He implemented the technique using a rule-based classifier, which was trained to predict the class labels of all hyperlinks that point to a page. A hyperlink ensemble was then formed by obtaining one prediction for each hyperlink. These individual predictions for each hyperlink pointing to the same page were subsequently combined to a final prediction for the class of the target page by looking for the majority among the individual predictions. He explored four different ways of combining the individual predictions and four different techniques for identifying relevant text portions. The best results were achieved by selecting the prediction with the maximum confidence from the ensemble. His experiments illustrated that the use of information about the HTML structure of pages and about the structure of the Web itself can be useful for improving text categorization on the Web. However, his conclusion was drawn from the off-line experiments on a single domain. How his techniques would perform in an open domain, i.e., for classifying arbitrary pages with arbitrary hyperlinks, remains an open question.

Uren and Addis (2002) explored a theoretical approach to predicting the performance of combined systems and used its predictions to benchmark the real results of a number of combined systems. Simple voting system was used in combining. They found that the actual accuracy achieved by combining text categorizers was considerably less than the accuracy that would be predicted if the errors produced by the systems were independent of each other. This performance conformed closely to the coincident errors model which assumes that some records are more likely to cause errors than others. Hence, typical categorization approaches produce predictions which are too similar for combining them to be effective since they tend to fail on the same records. The coincident errors could be reduced if a categorizer using a different kind of information as input was included in the combined system. Further experiments suggested that combining text categorizers can be successful, provided the essential element of difference is considered.

Chen and Ho (2000) studied the performance of decision forests where multiple decision trees are constructed systematically by pseudo-randomly selecting subsets of components of the feature vector. The classifier maximizes an average of the estimates of posterior probabilities given by individual trees. It was demonstrated that, based on micro-average recall and F_1 measure, the method outperformed single decision trees by *C4.5* and k-NN classifiers that were constructed using all the available features.

5.2 Dempster-Shafer's Evidence Combination

A popular approach to combining decisions from several sources is the fusion that uses the cornerstone of the Dempster-Shafer theory known as the Dempster-Shafer rule of evidence combination (Shafer 1976). Dempster-Shafer theory is a generalization of the Bayesian theory of subjective probability. It offers an alternative to traditional probabilistic theory for representing uncertainties and lack of knowledge. Many researchers have applied DST to the problem of classifier combination since there is usually uncertainty associated with the performance of each of the classifiers and DST has the ability to represent this uncertain knowledge and imprecision embedded in evidence. DST is an effective tool for combining data and knowledge from heterogeneous sources in the presence of conflicting information and it does not require any assumption regarding the probability of the individual constituents of the classifier set. Unlike the Bayesian method, DST allows user to specify a degree of ignorance when user has limited or unknown information instead of being forced to supply some probabilities which add to unity.

Consider a finite set of mutually exclusive and exhaustive propositions, $\Theta = \{\theta_1, \ldots, \theta_k\}$, referred to as the *frame of discernment* (FoD). In our context, θ_i states that "document x belongs to category θ_i." Any subset $A \subseteq \Theta$ also represents a proposition. The essence of DST is in the *Basic Belief Assignment* (BBA) that assigns to any $A \subseteq \Theta$ a numeric value $m(A) \in [0, 1]$ that satisfies the following conditions (Shafer 1976):

$$m(\emptyset) \quad = \quad 0 \tag{5.1}$$

$$\sum_{A \subseteq \Theta} m(A) \quad = \quad 1 \tag{5.2}$$

This so-called *mass function*, $m(A)$, quantifies the strength of evidence that supports proposition A. Proposition A such that $m(A) > 0$ is called a focal element of Θ, and the set of all focal elements of Θ is $\mathfrak{F}(\Theta) = \{A \subseteq \Theta : m(A) > 0\}$. This non-classical idea of "mass" in DST is different from probability in Bayes theory, although the two measures look very similar. As can be seen from Equation 5.2, if $\bar{A}$ is the complementary set of A, then $m(A) + m(\bar{A}) \leq 1$.

The *belief function* assigns to every nonempty subset $A \subseteq \Theta$ a value $Bel(A) \in [0, 1]$ that is called "degree of belief in A." It is defined as follows:

$$Bel(A) \quad = \quad \sum_{B \subseteq A} m(B) \tag{5.3}$$

This means that the strength of belief in proposition A is the sum of the strengths of beliefs in all subsets of A. The belief assigned to A thus takes into account the supports of all proper subsets of A. Note that $Bel(A) = m(A)$ if A is a singleton.

Dempster-Shafer rule of combination makes it possible to arrive at a new BBA by fusing the information from several BBAs that span the same FoD. Assume there are two bodies of evidence (BoE) $\{\Theta, \mathfrak{F}_1, m_1\}$ and $\{\Theta, \mathfrak{F}_2, m_2\}$. That is, m_1 and m_2 are BBAs for the same FoD Θ with focal elements $\mathfrak{F}_1$ and $\mathfrak{F}_2$, respectively. The normalization constant,

$$K_{12} \quad = \quad 1 - \sum_{\substack{B_i \in \mathfrak{F}_1; C_j \in \mathfrak{F}_2; \\ B_i \cap C_j = \emptyset}} m_1(B_i) m_2(C_j),$$

measures how much m_1 and m_2 are conflicting. If $K_{12} > 0$, the two BoEs are said to be compatible, and m_1 and m_2 can be combined to give m for any $A \neq \emptyset$ by Dempster's evidence combination function (DECF):

$$\begin{aligned} m(A) &\equiv (m_1 \oplus m_2)(A) \\ &= \sum_{\substack{B_i \in \mathfrak{F}_1; C_j \in \mathfrak{F}_2; \\ B_i \cap C_j = A}} m_1(B_i) m_2(C_j) \div K_{12}. \end{aligned} \tag{5.4}$$

Among previous attempts to apply DST to classical single-label classification problems, Bahler and Navarro (2000) compared five ensemble methods (DST, majority voting, Bayesian classification, behavior-knowledge space, and logistic regression) that allow the combination of classifiers obtained by different learning paradigms. They observed an improvement in the performance of DST-based combination when the members of the ensemble have non-zero rejection rates. Al-Ani and Deriche (2002) discussed existing methods for computing evidence and described a new classifier combination technique based on DST that adapts to training data so as to minimize the overall mean square error. The technique outperformed other classifier combination methods on three different domains.

Bi et al. (2004) investigated the combination of four different classification methods for text categorization, SVM, k-NN, k-NN model-based approach, and Rocchio methods. They employed a 2-points focused mass function to represent outputs from these different classifiers based on the confidence values for labels. The experimental results showed that the performance of the best combination of the different classifiers on ten benchmark domains slightly outperformed the best individual method. Another closely related method based on Dempster's rule for combination of evidence was described by Bell et al. (2005). They developed and analyzed an evidence structure for representing outputs from different classifiers using mass functions with a focal element triplet or a focal element quartet. Bi et al. (2005) proposed a boosting-like technique for generating multiple sets of rules based on rough set theory and model classification decisions from multiple sets of rules as pieces of evidence which can be combined by Dempsters rule of combination. Similarly, their experiments with the 20-newsgroup which is a public benchmark data collection indicated that the performance of the best combination of the multiple sets of rules outperformed the best single set of rules. Furthermore, the comparative analysis between the Dempster-Shafer and the majority voting methods confirmed the advantage and the robustness of their approach.

Altınçay (2005) proposed a dynamic integration of classifier ensembles that involve members trained on heterogeneous input sets. The approach is based on an evidence-theoretic framework that takes into account the weights and distances of the neighboring training examples in boosting ensembles.

These DST-approach papers only deal with the single label case where each instance belongs to exactly one category. Combining classifier results when the problem is multi-label has not received adequate attention.

5.3 Dempster-Shafer Fusion for Multi-label Case

As shown in Figure 4.4 of Chapter 4, the proposed mechanism for multi-label classifier induction requires a master algorithm to fuse the recommendations from several subclassifiers that are induced by a baseline learning algorithm. A choice of baseline algorithm has been discussed and *AdaBoost.MH* is chosen because it runs reasonably fast and can produce outputs in the form easy for fusion. Moreover, from the experiments, it outperformed other candidates. For the master algorithm, a number of existing classifier fusion methods could be implemented in our classifier induction system after they are modified to handle the multi-label case. The possibility of employing them will be explored in Chapter 7. This section presents a new fusion technique that will be used as the master algorithm. Taking on the Dempster-Shafer belief theoretic framework, the technique is derived to have an added capability of combining outputs from multi-label classifiers. Specifically, DECF (Equation (5.4)) is used to fuse multi-label outputs generated from subclassifiers resulting from the application of *AdaBoost.MH* on data of different feature subsets.

We choose to develop a master combination algorithm based on DST mainly because: 1) each small data set tends to produce different class labels for an example and DST is efficacious in combining conflicting information and 2) DST can manage uncertainty without making any unrealistic probability assumption. It also appears that DST-approach has a satisfactory performance in single-label classification problems. There have been some attempts to apply DST to multi-label problems by focusing on a certain subset of labels and transforming the problems into binary classification problems. However, no literature on the application of DST to a real multi-label problem has been found so far.

Evidence from each classifier plays a crucial role in the fusion task since it influences the performance of the fused result. The issue here is how to properly quantify the evidence of subclassifiers. Given an example x, the baseline algorithm, *AdaBoost.MH*, provides the ranking function $f(x, .)$ along with other classification results. $f(x, .)$ is a measure of confidence in each label that is obtained from the application of *AdaBoost.MH* to each of feature subsets. To exploit the confidence information from subclassifiers in the fusion process, we develop a method that transforms the confidence measure into belief theoretic information, on which the DECF will then be applied.

Figure 5.1 displays the architecture of the complete multi-label classifier induction system. It is Figure 4.4 with details of the fusion step inserted. The training data set goes through usual data preprocessing and feature selection process and the feature set is divided into N feature subsets. Then apply the baseline learning algorithm to induce subclassifiers from feature subsets. Outputs of subclassifiers, $f(.,.)$ are transformed into basic belief assignment and finally are combined into the overall belief information $m(.)$ to be used by the master classifier.

Outlined in Figure 5.2 is the classification procedure for a new example x. Each time an example x is to be categorized, the j-th subclassifier outputs the ranking function, $f_j(x,.)$, that reflects the confidence of the subclassifier in each label for x. The algorithm will convert $f_j(x,.)$ into $m_j(.)$ for all j. Then all $m_j(.)$'s are combined into $m(.)$ from which a set, $Y \subseteq \mathcal{Y}$ is output as the example's class labels .

Next section describes the computation of belief functions from classifier outputs. A detailed explanation of fusion step is presented with a rigorous derivation of required formulas.

5.3.1 Basic belief assignment in multi-label case

Learning algorithms that are tailored specially for multi-class, multi-label classification problems usually can offer soft decisions about categories in addition to hard decisions, a boosting algorithm like *AdaBoost.MH* included. These algorithms give rise to a set of decision rules which produce classification results of an example x in the form of a ranking function or confidence measure given to each class label c, $f(x,c)$. The sign of the prediction $f(x,c)$ indicates whether the label c is assigned to the example x, while the magnitude of $|f(x,c)|$ signifies the confidence level on the label. More precisely, the example is categorized as being in category c if $f(x,c)$ is positive, and not in category c otherwise. The value of $|f(x,c)|$ provides evidence

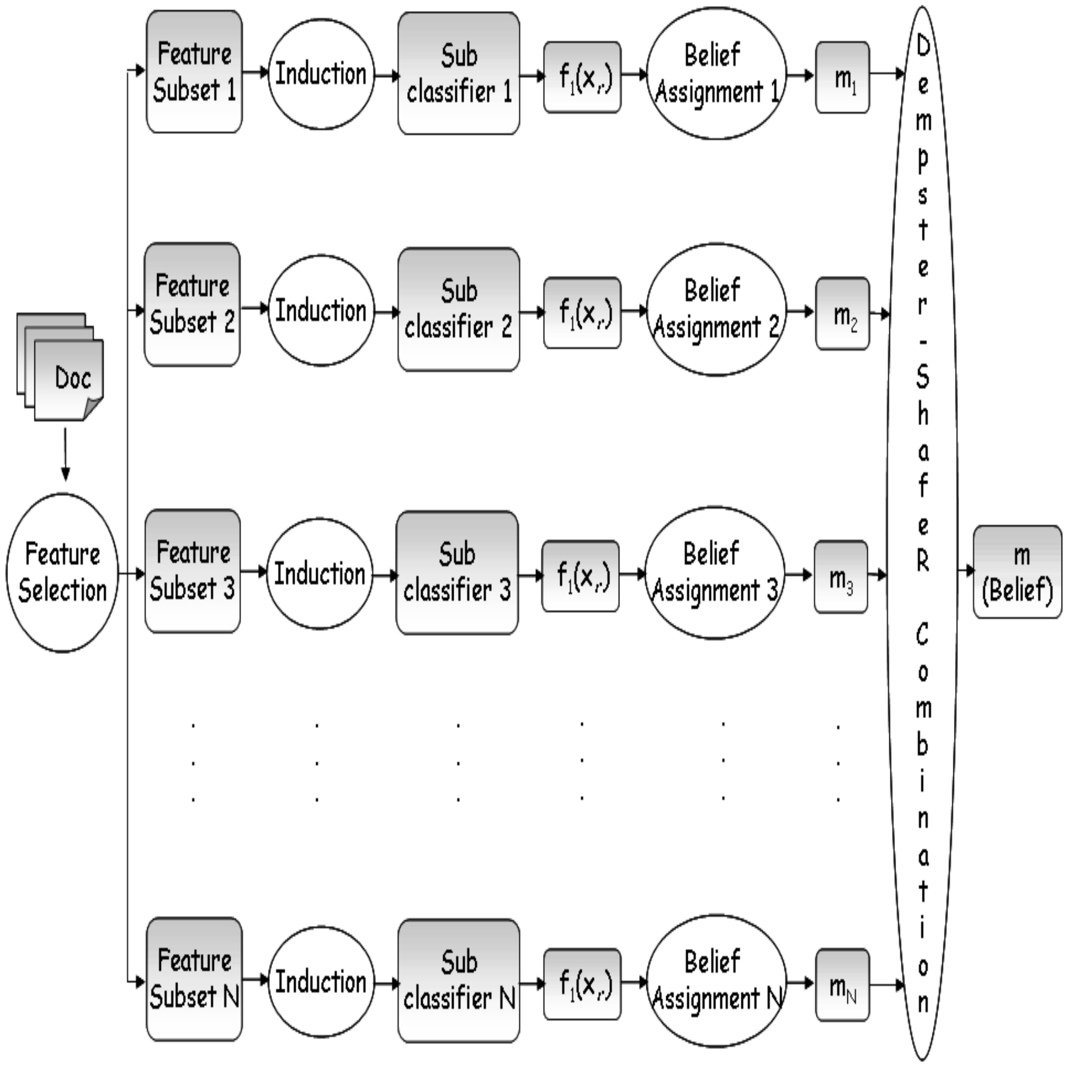


Figure 5.1: Classifier induction system where a baseline induction algorithm is run on different feature subsets to generate subclassifiers and a master algorithm uses the Dempster-Shafer theory to combine subclassifiers' evidence.

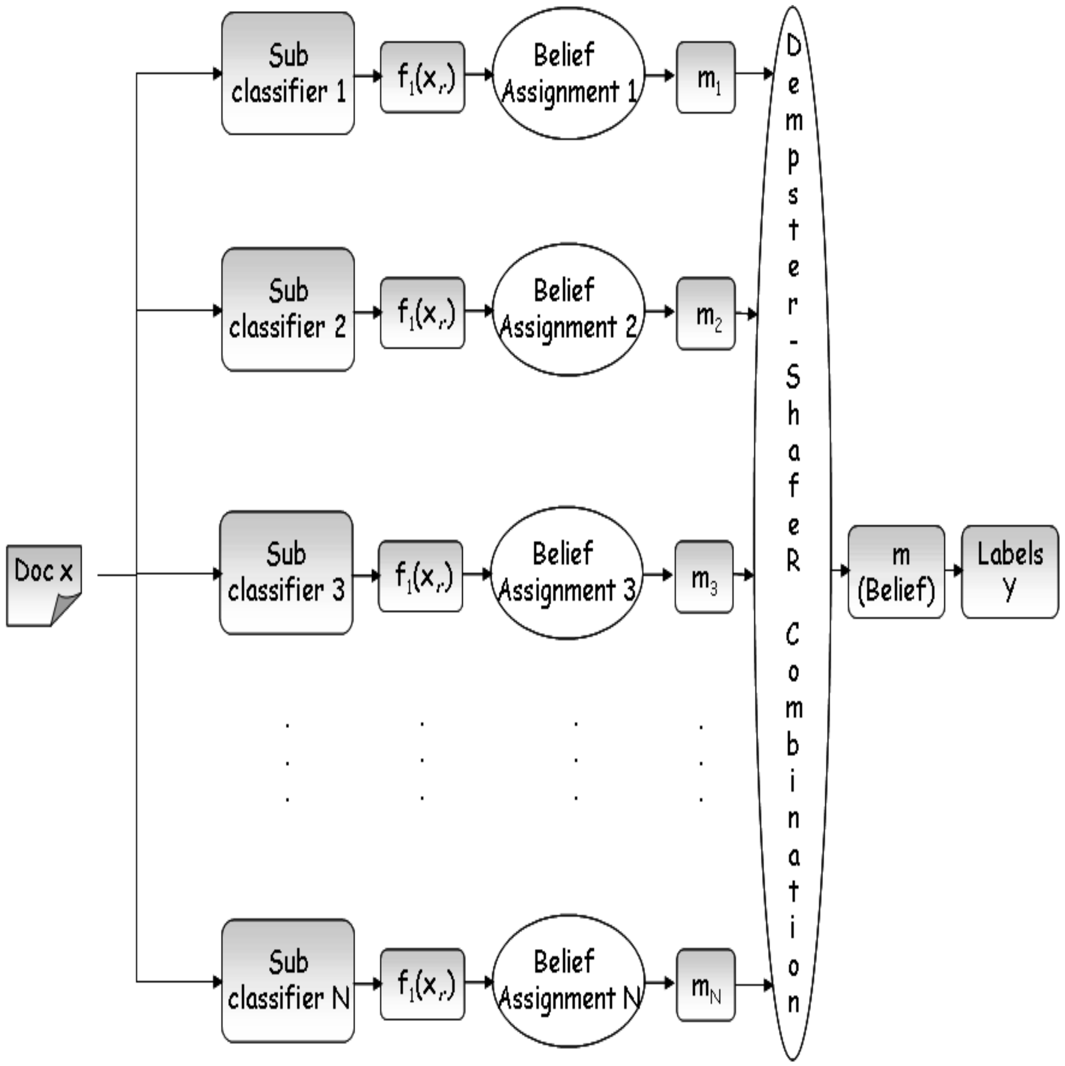


Figure 5.2: Classification of an example where a master classifier uses the Dempster-Shafer theory to combine the “testimonies” of a group of subclassifiers.

either supporting or against the class label c, the higher value of $|f(x,c)|$ meaning the more confidence or stronger belief in label c if $f(x,c)$ is positive, and stronger disbelief in label c if $f(x,c)$ is negative.

The confidence $f(.,.)$ on each label is a real number, $f(.,.) \in \mathbb{R}$, that can be converted to a degree of belief with values between 0 and 1. It should be noted that the strong belief in one label of an example does not negate nor enhance the possibility of the example having other labels. Thus as long as after the conversion labels are kept in the same order as before, we can attend to just one label at a time and combine the belief values corresponding to that label from all classifiers.

For each classifier, follow the two major transformation steps detailed below to assign basic belief to a label c.

1. Normalize the confidence measure $f(.,.)$ of labels such that it lies between 0 and 1.

   Let $f^*(x,c)$ be the normalized confidence level of label c for an example x, $f^*(x,c) \in [0,1]$. $f^*(x,c)$ is treated as a degree of belief. The value of $f^*(x,c)$ close to 1 is a strong belief in the label c, the value close to 0 is the strong disbelief in label c, and the value close to 0.5 is the indecisive stage. Therefore we distinguish 3 cases in the conversion of $f(x,c)$ to $f^*(x,c)$.

   (a) When $f(x,c) \geq 0$ for every label, all $f(x,c)$'s will be transformed to lie in the range $[0.5,1]$ with the minimum of $f(x,c)$ being 0.5 and the maximum of $f(x,c)$ being 1.
   $$f^*(x,c) = 0.5 + \frac{f(x,c) - \min\limits_{c'} f(x,c')}{2 \times (\max\limits_{c'} f(x,c') - \min\limits_{c'} f(x,c'))} \tag{5.5}$$

   (b) When $f(x,c) < 0$ for every label, all $f(x,c)$'s will be transformed to stay in the range $[0,0.5]$ with the minimum of $f(x,c)$ being 0 and the maximum of $f(x,c)$ being 0.5.

$$f^*(x,c) = \frac{f(x,c) - \min_{c'} f(x,c')}{2 \times (\max_{c'} f(x,c') - \min_{c'} f(x,c'))} \tag{5.6}$$

(c) When some $f(x,c) \geq 0$ and some $f(x,c) < 0$, the transformation will be asymmetric. All negative $f(x,c)$'s are transformed to the range $[0, 0.5)$ with the minimum of $f(x,c)$ being 0. All non-negative $f(x,c)$'s are transformed to the range $[0.5, 1]$ with the maximum of $f(x,c)$ being 1. This is to keep the interpretation of the values of $f^*(x,c)$ similar to that of $f(x,c)$; $f(x,c) \geq 0$ or equivalently, $f^*(x,c) \geq 0.5$ indicates the label c should be assigned to an example x, and vice versa.

$$f^*(x,c) = \begin{cases} -\frac{f(x,c) - \min_{c'} f(x,c')}{2 \times \min_{c'} f(x,c')} \;, & \text{if } f(x,c) < 0 \\ 0.5 + \frac{f(x,c)}{2 \times \max_{c'} f(x,c')} \;, & \text{if } f(x,c) \geq 0 \end{cases} \tag{5.7}$$

2. Compute masses or BBA for each label.

Define $P(c|c')$ to be the conditional probability of assigning class label c to a class c' example. When $c \neq c'$, $P(c|c')$ is simply the misclassification probability of the classifier. Let $\bar{c}$ denote the event that the example does not have label c. Then

$$P(c|\bar{c}) + P(\bar{c}|\bar{c}) \quad = \quad 1$$

and

$$P(c|c) + P(\bar{c}|c) \quad = \quad 1.$$

Given a set of learning examples, $P(c|\bar{c})$, $P(\bar{c}|\bar{c})$, $P(c|c)$, and $P(\bar{c}|c)$ for any class c can be estimated as follows.

$$P(c|\bar{c}) \quad = \quad \frac{\#\text{of examples not in class } c \text{ that are classified as class } c}{\#\text{of examples that are not class } c}$$

$$P(\bar{c}|c) \quad = \quad \frac{\#\text{of class } c \text{ examples that are not classified as class } c}{\#\text{of class } c \text{ examples}}$$

Accordingly,

$$P(\bar{c}|\bar{c}) \quad = \quad 1 - P(c|\bar{c})$$

and

$$P(c|c) \quad = \quad 1 - P(\bar{c}|c).$$

$P(c|\bar{c})$ and $P(\bar{c}|c)$ are classification errors which tell us how accurate the classifier is. These two quantities play an important role when many pieces of evidence from multiple classifiers are to be combined since they imply some level of uncertainty in the information provided by classifiers.

Denote the prior probability that an example is in class c as $P(c)$ and the prior probability that a document is not in class c as $P(\bar{c})$.

$$P(c) + P(\bar{c}) \; = \; 1$$

These probabilities can be estimated from learning examples.

$$P(c) \; = \; \frac{\#\text{of class } c \text{ examples}}{\text{Total}\#\text{of examples}} \tag{5.8}$$

$$P(\bar{c}) \; = \; 1 - P(c)$$

The credibility of a classifier depends on its prediction accuracy. How well a classifier is in making a decision about the class c membership of an example is determined by the sum of two errors, $P(c|\bar{c})P(\bar{c}) + P(\bar{c}|c)P(c)$. The first error is the case when an example not in class c is incorrectly labeled as class c, and the second error is the case when a class c example is not recognized as such. Naturally, when combining the evidence that supports class c from multiple classifiers, we should take into account the performance pertinent to class c of each classifier. Therefore, classifiers that have a larger sum of errors when categorizing class c examples will receive smaller weight.

Let $\Theta = \{c, \bar{c}\}$ be the frame of discernment (FoD) or sample space when considering a fixed label c. The BBA $m(.)$ for FoD Θ must satisfy the conditions in (5.1) and (5.2). A set of focal elements of Θ is $\mathfrak{F}(\Theta) = \{A \subseteq \Theta : m(A) > 0\} = \{c, \bar{c}, \Theta\}$. $m(\{c\})$ and $m(\{\bar{c}\})$ are the supports assigned to the proposition that an example has label c and the proposition that an example does not have label c, respectively. For simplicity, we will write $m(c)$ instead of $m(\{c\})$ and $m(\bar{c})$ instead of $m(\{\bar{c}\})$.

When assigning the values to $m(c)$ and $m(\bar{c})$, both the confidence in the label c and the accuracy of the classifier in choosing class c must be taken into consideration. The confidence in the label c is given by $f^*(x, c)$ and the class c classification accuracy of the classifier is

$$Acc(c) = P(c|c)P(c) + P(\bar{c}|\bar{c})P(\bar{c}) \tag{5.9}$$

or, equivalently,

$$Acc(c) = 1 - \{P(c|\bar{c})P(\bar{c}) + P(\bar{c}|c)P(c)\}.$$

Θ represents the event of imperfect knowledge regarding the class labels of the example. In other words, we are in doubt whether the document has label c or not. This is perhaps due to the lack of trust in the classifier, and consequently the classification result provided by the classifier should be ignored. From this rationale, the BBA assigned to the FoD Θ is calculated as:

$$m(\Theta) = P(c|\bar{c})P(\bar{c}) + P(\bar{c}|c)P(c) \tag{5.10}$$

$$m(c) = f^*(x, c) * \{1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)\} \tag{5.11}$$

$$\begin{aligned} m(\bar{c}) &= f^*(x, \bar{c}) * \{1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)\} \\ &= (1 - f^*(x, c)) * \{1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)\} \end{aligned} \tag{5.12}$$

The computation of basic belief assignments just described is summarized in Figure 5.3. Appendix A gives the formal proof that the masses thus calculated satisfy the condition from Equation 5.2.

5.3.2 Combining evidence from multiple classifiers

To decide on the labels of a new example, one must gather information regarding the belief in each label from all subclassifiers. Individual subclassifiers may produce different decisions, but the conflicts can be resolved to some extent by DST. A group decision will be made by new classification criteria derived from the summative information of the subclassifier committees. Let us start with two subclassifiers. Any two items of evidence favoring or against a class label from two subclassifiers can be combined into one piece by Dempster's rule as follows.

Let m_1 and m_2 be two sets of BBAs associated with a label c for the same frame of discernment Θ that correspond to subclassifiers 1 and 2, respectively. Let m denote the BBA resulting from the combination of m_1 and m_2.

Consider the normalization constant

$$K_{12} \quad = \quad 1 - \{m_1(c)m_2(\bar{c}) + m_1(\bar{c})m_2(c)\}.$$

K_{12} measures the conflicting information about the label c from the two subclassifiers. Providing $K_{12} > 0$, the two bodies of evidence $\{\Theta, \mathfrak{F}, m_1\}$ and $\{\Theta, \mathfrak{F}, m_2\}$ are compatible. Then Dempster's evidence combination function m is

Steps in assigning basic belief to each label:

1. For each classifier, compute the degree of belief, $f^*(x,c)$, in each label.

   (a) When $f(x,c) \geq 0$ for all labels,

   $$f^*(x,c) = 0.5 + \frac{f(x,c) - \min_{c'} f(x,c')}{2 \times [\max_{c'} f(x,c') - \min_{c'} f(x,c')]}$$

   (b) When $f(x,c) < 0$ for all labels,

   $$f^*(x,c) = \frac{f(x,c) - \min_{c'} f(x,c')}{2 \times [\max_{c'} f(x,c') - \min_{c'} f(x,c')]}$$

   (c) When some $f(x,c) \geq 0$ and some $f(x,c) < 0$,

   $$f^*(x,c) = \begin{cases} -\frac{f(x,c) - \min_{c'} f(x,c')}{2 \times \min_{c'} f(x,c')}, & \text{if } f(x,c) < 0 \\ 0.5 + \frac{f(x,c)}{2 \times \max_{c'} f(x,c')}, & \text{if } f(x,c) \geq 0 \end{cases}$$

2. For each classifier, compute the BBA for each label as follows.

   (a) Compute classification error probabilities for each label:

   $$P(\bar{c}|c) \quad = \quad \frac{\#\text{of class } c \text{ examples that are not classified as class } c}{\#\text{of class } c \text{ examples}}$$

   $$P(c|\bar{c}) = \frac{\#\text{of examples not in class } c \text{ that are classified as class } c}{\#\text{of examples that are not class } c}$$

   (b) Estimate prior probability of each label:

   $$P(c) \;=\; \frac{\#\text{of class } c \text{ examples}}{\text{Total } \#\text{of examples}} \text{ and } P(\bar{c}) \;=\; 1 - P(c)$$

   (c) Compute a set of BBAs associated with each label:

   $$m(\Theta) \;=\; P(c|\bar{c})P(\bar{c}) + P(\bar{c}|c)P(c)$$

   $$m(c) \;=\; f^*(x,c) * \{1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)\}$$

   $$\begin{aligned} m(\bar{c}) &= f^*(x,\bar{c}) * \{1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)\} \\ &= (1 - f^*(x,c)) * \{1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)\} \end{aligned}$$

Figure 5.3: Computing basic belief assignments in multi-label problems

$$
\begin{aligned}
m(A) &\equiv (m_1 \oplus m_2)(A) \\
&= \begin{cases}
0, & \text{if } A = \emptyset \\
(m_1(c)m_2(c) + m_1(c)m_2(\Theta) + m_1(\Theta)m_2(c)) \div K_{12}, & \text{if } A = c \\
(m_1(\bar{c})m_2(\bar{c}) + m_1(\bar{c})m_2(\Theta) + m_1(\Theta)m_2(\bar{c})) \div K_{12}, & \text{if } A = \bar{c} \\
(m_1(\Theta)m_2(\Theta)) \div K_{12}, & \text{if } A = \Theta
\end{cases}
\end{aligned}
\tag{5.13}
$$

The orthogonal sum operator $\oplus$ possesses the commutative and associative properties:

$$(m_1 \oplus m_2)(A) = (m_2 \oplus m_1)(A) \tag{5.14}$$

$$m_1 \oplus (m_2 \oplus m_3)(A) = (m_1 \oplus m_2) \oplus m_3(A), \tag{5.15}$$

Thus, the above evidence combination function (5.13) that are used to combine two subclassifiers can be extended to the combination of any number of subclassifiers in a straightforward manner.

Since there are only two elements, c and $\bar{c}$, in the frame Θ, the beliefs assigned to singletons c and $\bar{c}$ are $Bel(c) = m(c)$ and $Bel(\bar{c}) = m(\bar{c})$. Therefore, after the fusion process is completely done, the final classification rule for predicting class c can be established as a simple decision rule:

"Assign label c to the new example if $Bel(c) > Bel(\bar{c})$."

If $Bel(c) \leq Bel(\bar{c})$, declare that the new example does not belong to class c.

CHAPTER 6

Performance of Multi-label Dempster-Shafer Fusion

Having proposed and presented a solution in the previous chapters to deal with a large number of features when inducing multi-label classifiers, it is necessary to get a clear idea on how the classifier induction system that was developed performs in practical applications. We proceed to implement the master algorithm, DST-Fusion, with *AdaBoost.MH* as a baseline learner and evaluate its performance empirically on two real-life data sets. We revisit the EUROVOC database and conduct an in-depth study on the system behavior. Some of the early experiments demonstrated a promising performance of the DST-Fusion on subsets of EUROVOC data (Sarinnapakorn and Kubat 2007b, 2007a, 2008). Results from a full set of experiments will be reported in this chapter. In particular, the performance of DST-Fusion is compared to the regular no fusion approach where the *AdaBoost.MH* is applied straight on the whole feature set. We focus on studying the effects of a variety of factors, e.g., the number of subclassifiers and the number of boosting rounds, on the DST-Fusion's performance. The experiments are designed with different parameter configurations; the general framework is described in the Experimental Setup section.

Experiments with EUROVOC data prove that DST Fusion is efficient and has acceptable predictive accuracy for text categorization task. Likewise, further experiments with another data set, documents from the Reuters test collection, confirm the same outcome.

6.1 Experimental Setup

Hundreds of experiments are carried out to answer several issues of interest. Three main factors that can affect the performance of DST-Fusion are examined at this point.

- Total number of features in the original feature set
- Number of feature subsets (or number of subclassifiers to be combined by the master algorithm)
- Number of boosting iterations used in the AdaBoost.MH

The number of boosting iterations is a parameter that must be specified when running *AdaBoost.MH*. It has an influence on the accuracy of *AdaBoost.MH*. In each iteration only one feature is selected for weak hypothesis. As such, the total number of features actually utilized in the *AdaBoost.MH* classifier is bounded by the number of boosting rounds. We are tempted to use a large number of boosting rounds to allow more features be selected. However, there seems to be a controversy as to whether AdaBoost will overfit the training data. Overfitting refers to the situation when, after going through the training cycle too many times, the training error stabilizes (does not decrease any more) or sometimes even starts to increase. It was shown theoretically and empirically by Grove and Schuurmans (1998) and Schapire (1999) that boosting overfits, whereas some other authors, Drucker and Cortes (1996), Quinlan (1996a), and Breiman (2001), observed empirically that boosting tends not to overfit, even when run for thousands of rounds. Thus concerning what number of boosting iterations to use in our study, we decided to run two different sets of experiments. In one setting, always use a fixed number of boosting rounds regardless of the number of features in the training data. And in the other, set the number of boosting rounds as a fixed percentage of the number of features in the training data.

When separating features in the original feature set into subsets, there is a question about how to form feature subsets. Previous studies of similar nature indicated that fusion of classifiers from feature subsets that are overlapping usually outperforms individual subclassifiers, and in many applications it is better than the classifier from all features. We too reach the same conclusion in our experimentation that overlapping feature subsets offer a slightly higher accuracy than non-overlapping ones. The advantage of overlapping over non-overlapping will be fully discussed later in Chapter 8. For almost all of the experiments discussed here, we use overlapping feature subsets, even though it will increase the computation time of fusion. In some instances, results from non-overlapping feature subsets will also be presented for comparison.

DST-Fusion is tested and compared to "NoFusion" approach on two data sets, a simplified EUROVOC database described in Section 4.1 and a publicly available data set in Reuters test collection, Reuters Corpus Volume 1 version 2 (RCV1-v2). RCV1 has been used as a benchmark for many supervised learning methods including Naive Bayes, SVM and k-NN. For NoFusion, AdaBoost.MH is run on all features and for DST-Fusion, the system runs AdaBoost.MH on feature subsets with subsequent fusion of their recommendations. Experiments in this chapter share the same basic design:

- No any special technique for feature selection is used.
- Feature subsets are of equal size and 20% of the features in each subset are overlapping with other subsets. Thus, if the feature set has 4,000 features and

five feature subsets are to be formed, then each feature subset will have 1,000 features, 200 of which also appear in other subsets.

- Evaluation criteria are based on those performance measures in Sectionse:perform and the computational costs which are measured by the total CPU time required for induction of classification rules and classification of test examples. The experiments are run on an AMD 64 bits Dual Core X2 3800+ with 3GB memory. Note that the total CPU time is governed by the induction process since the classification of test examples actually takes only a small fraction of induction time.

- For EUROVOC data, the performance measures are estimated from 5-fold CV. For RCV1-v2 data, 5 independent training sets and 5 independent test sets are readily available [1] (Lewis et al. 2004), so there is no need for cross validation. The performance measures are estimated from 5 test data sets.

- Any statistical tests are based on 5% significance level.

6.2 Experiments with EUROVOC Data

In this section we study the specification of parameters involved in DST-Fusion. We carry out experiments on DST-Fusion to analyze the effects of three factors: number of features, number of subclassifiers, and number of boosting rounds. In the experiments, one factor is changed, the rest of them are fixed each time, and the impact on classification performance from the selected factor is studied. The results are presented below.

[1] http://mlkd.csd.auth.gr/multilabel.html#Datasets

6.2.1 Effect of number of features

In our first experiment, we randomly select 500, 1,000, 2,000, and 4,000 (or all) of the features from the simplified database. In each of the four cases, we divide the feature set into five equally-sized subsets with 20% overlapping with other subsets, thus obtaining sets of 125, 250, 500, and 1,000 features, respectively. The number of boosting rounds used by AdaBoost.MH is set at 10% of the number of features (e.g., 50 rounds for the experiment with 500 features).

Figures 6.1 and 6.2 compare the micro- and macro-averaging measurements of the DST-Fusion to NoFusion. DST-Fusion has lower micro-average precision but higher micro-average recall than NoFusion. Trading off between precision and recall, DST-Fusion becomes comparable to the NoFusion on micro-average F_1 basis. On the macro-averaging version, NoFusion wins DST-Fusion noticeably. DST-Fusion is as good as NoFusion on macro-average recall only when the number of features is 4,000. However, the low precision of DST-Fusion causes its F_1 measure to be lower. So, in general DST-Fusion does worse than NoFusion on rare or small classes. Also shown in the figures of micro-average F_1 and macro-average F_1 are the measurements of individual subclassifiers. DST-Fusion clearly outperforms each individual subclassifier. For other performance criteria specific to multi-label problems, we see from Figure 6.3 that NoFusion performs better than DST-Fusion. Hamming loss is not shown in this figure because it has similar pattern as the ranking loss. Besides, the difference between two methods is very small; yet it is significant. Complete results can be found in Appendix B which again support that DST-Fusion performs no worse than individual subclassifiers.

It is obvious that the performance of both systems improves with the growing number of features but the improvement is quite unimpressive—many of the thousands of features seem to be redundant or irrelevant. Interestingly, the margin between the

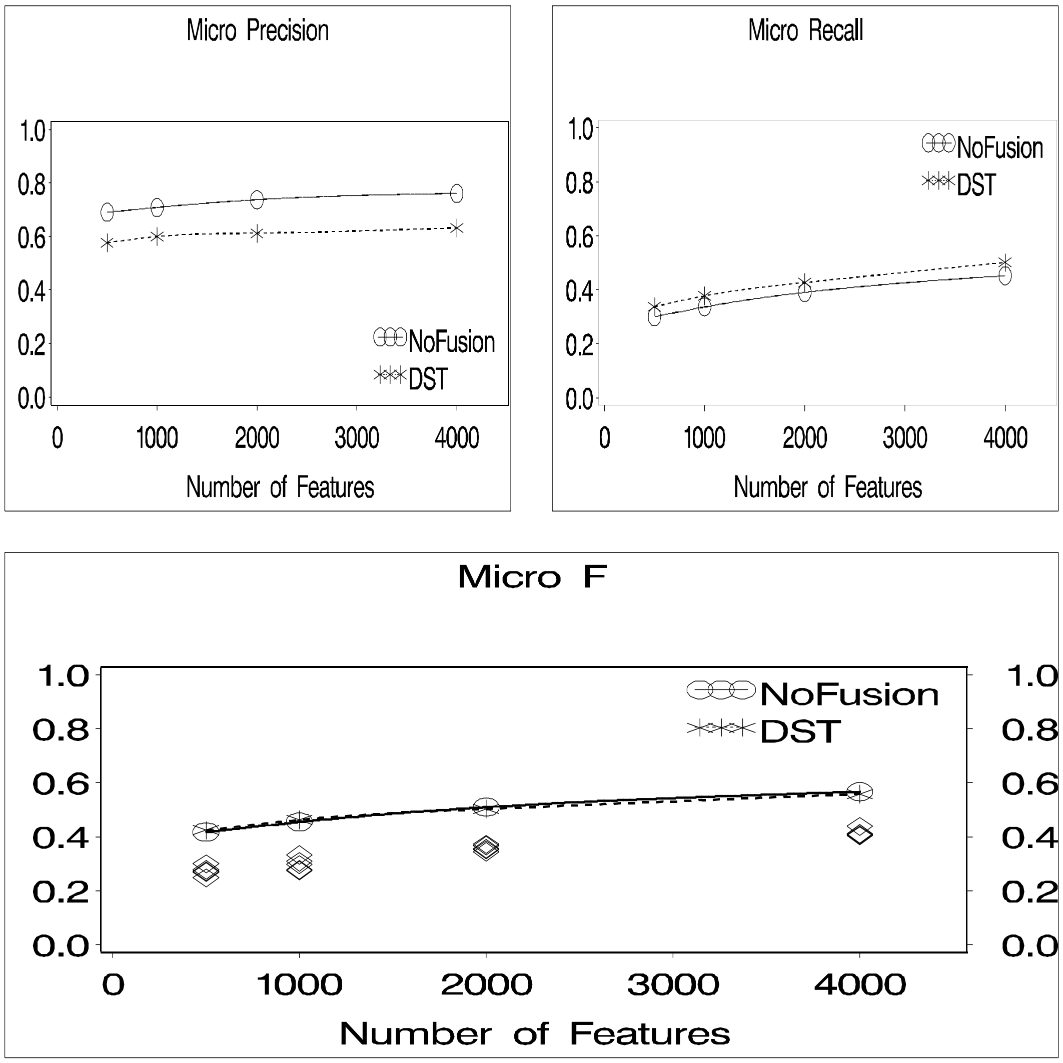


Figure 6.1: Micro-averaging of precision, recall, and F_1 measure of DST-Fusion and NoFusion when varying number of features. $\diamondsuit$ indicates F_1 of each subclassifier.

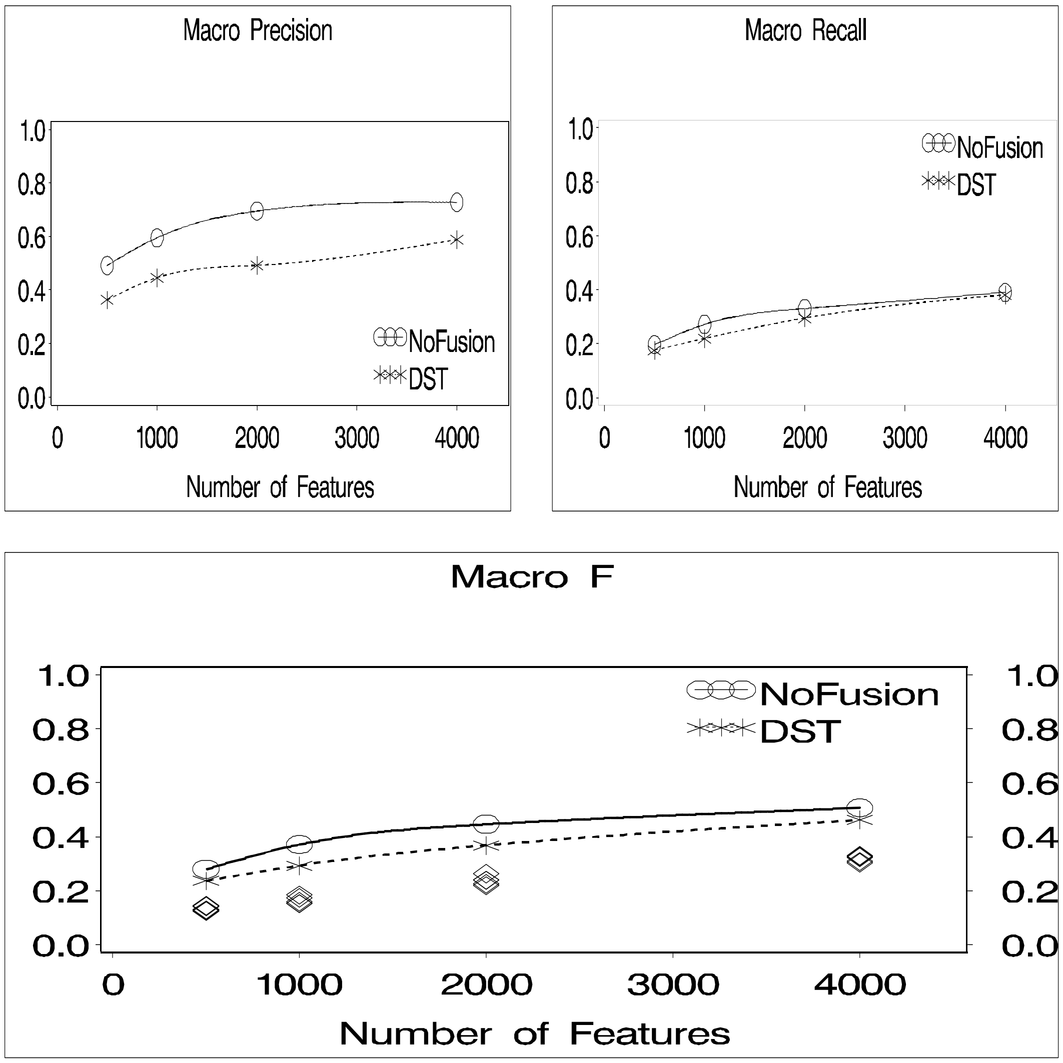


Figure 6.2: Macro-averaging of precision, recall, and F_1 measure of DST-Fusion and NoFusion when varying number of features. $\diamondsuit$ indicates F_1 of each subclassifier.

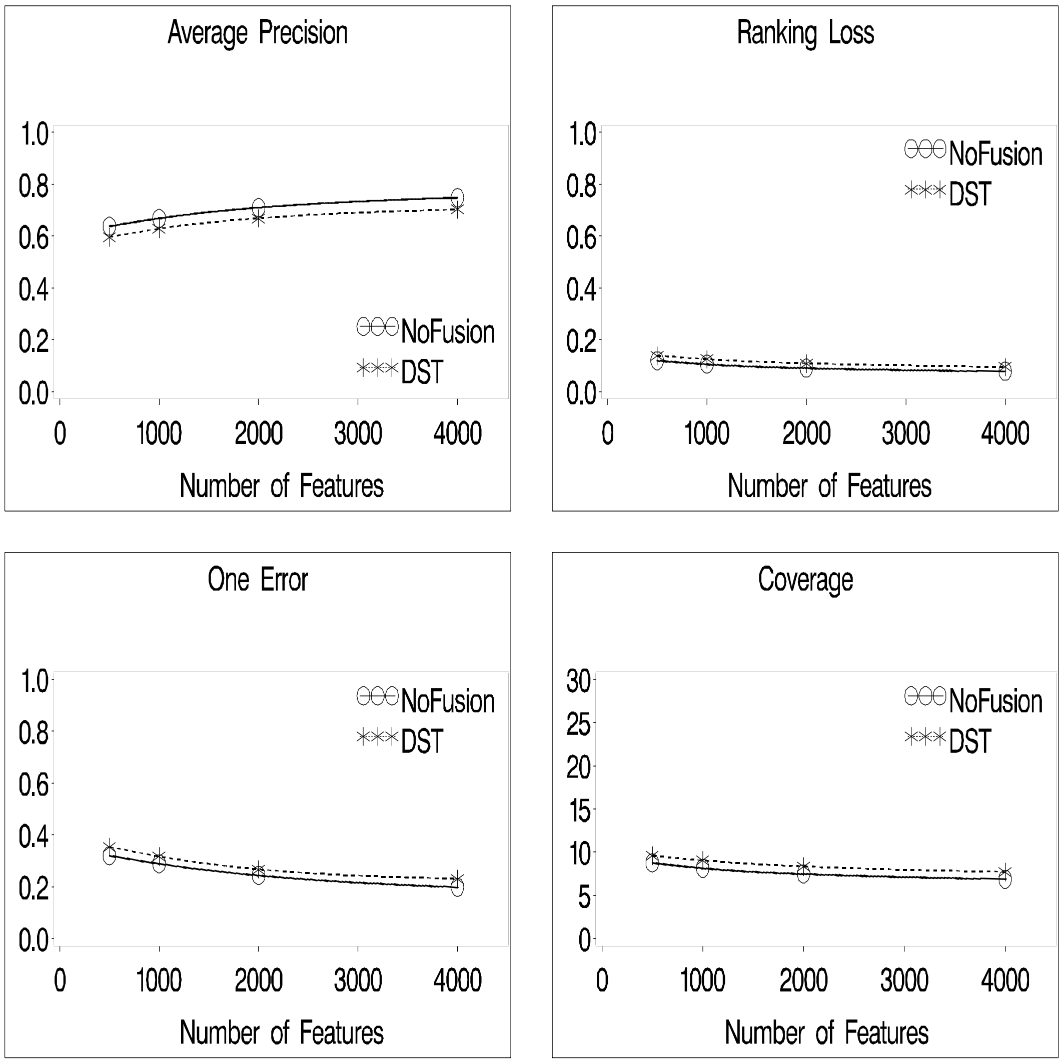


Figure 6.3: Multi-label performance measures of DST-Fusion and the NoFusion, when varying number of features.

performance of DST-Fusion and NoFusion does not appear to depend on the total number of features involved in the experiment. Furthermore, it is seen that classifier fusion always incurs certain loss in classification performance, e.g., lower average precision and higher ranking loss. Nonetheless, the loss never exceeds 5% on the average precision measure and 2% on the ranking loss. There is an exception with the micro-average F_1 measure where the DST-Fusion and NoFusion do not differ. In any case, these small losses are only marginal and may be deemed acceptable in view of the fact that features were chosen at random and no effort has been made to optimize feature-set selection. They can likely be tolerated if compensated by compelling computational savings.

Figure 6.4 depicts the interplay between the CPU time of induction and average precision. As we have seen earlier, the average precision increases steadily, though gradually, as we have more and more features. And while the difference in the average precisions of the two methods is relatively constant, the computational costs of NoFusion grow much faster than those of DST-Fusion. For example, for the case of 4,000 features, the induction of the NoFusion version takes more than five times longer than the induction of the DST-Fusion version. (Recall that for fusion we use feature subsets having 20% of the features overlapped with each other. This amounts to longer induction time to process extra number of features. If non-overlapping feature subsets are used or the overlapping percentage is smaller, DST-Fusion would use less time than what is shown here.) In the real-world setting, with a hundred thousand features in the entire domain of EUROVOC data, we expect this favorable impact to be even more strongly pronounced.

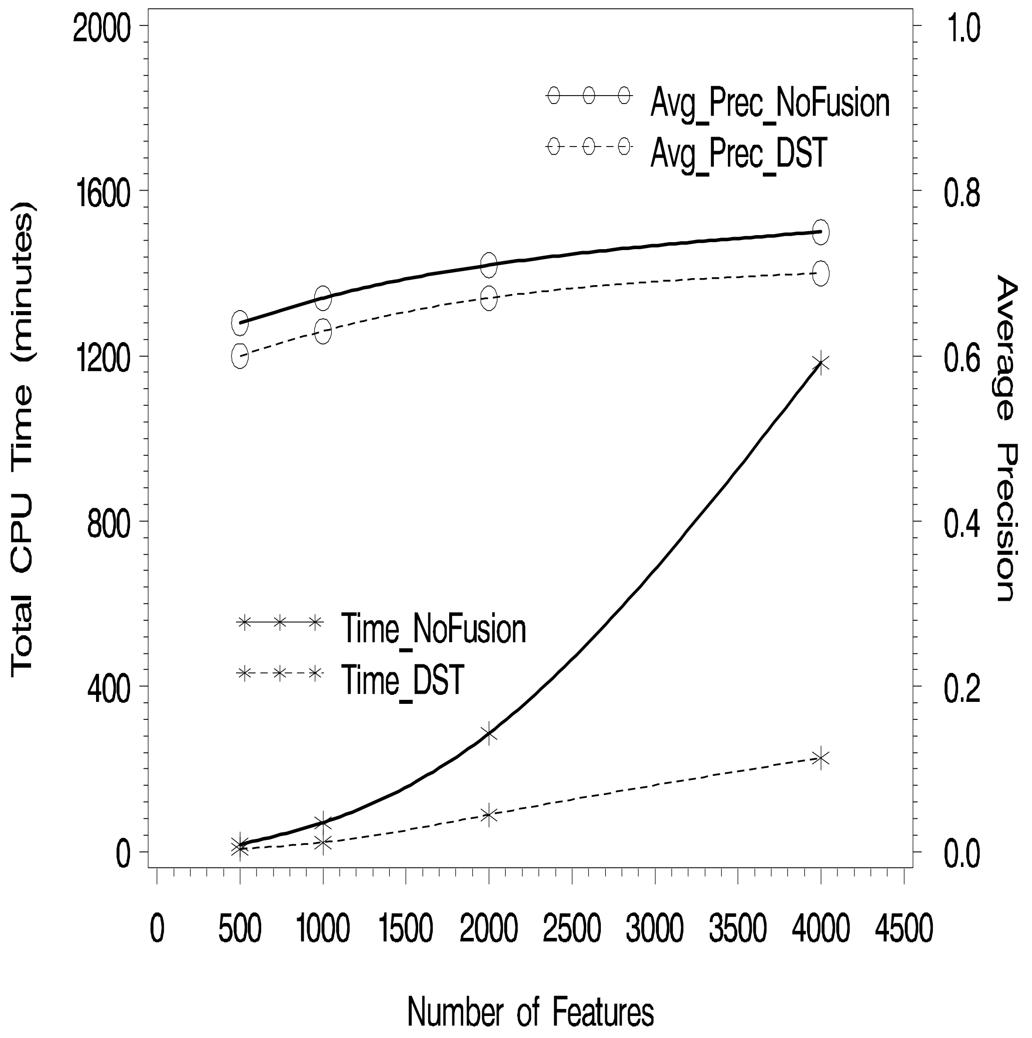


Figure 6.4: Total CPU time and average precision of DST-Fusion and NoFusion when varying number of features, 5-fold CV.

6.2.2 Effect of number of subclassifiers

In this section we want to understand how the behavior of the fusion system depends on the number of subclassifiers that it combines. With this aim, we divide the 4,000 features in four different ways such that each subset has 20% of the features overlapping with other sets: 5 sets of 1,000 features each, 10 sets of 500 features each, 20 sets of 250 features each, and 40 sets of 125 features each. For each of these divisions, we use AdaBoost.MH to induce subclassifiers from feature subsets and followed by DST-Fusion. As before, the number of boosting rounds is 10% of the number of features in the training set. The results of the DST-Fusion will be compared to the performance of "NoFusion" classifier which is a single classifier induced from all 4,000 features. We may think of this "NoFusion" classifier as a special case of DST-Fusion where the number of subclassifiers is 1.

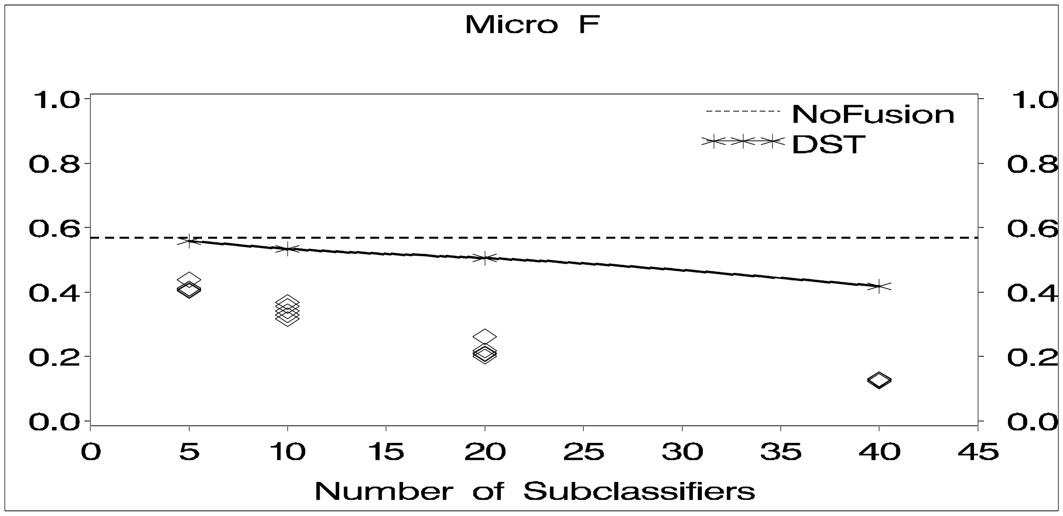
The results of all experiments are summarized in Figure 6.5 and Table 6.1. It is not surprising to observe that the number of subclassifiers has an important impact on the accuracy. The performance of DST-Fusion along all evaluation criteria deteriorates with the growing number of subclassifiers. Moving from the NoFusion case (or DST-Fusion of one subclassifier) to the DST-Fusion of 40 subclassifiers has led to a drop in average precision of 24%, from 75% to 51%, which is prohibitive. This deficiency notwithstanding, DST-Fusion is superior to individual subclassifiers. Another thing to notice is that the coverage goes up slowly from 6.89 of NoFusion to 8.93 of DST-Fusion of 20 subclassifiers, and then jumps to 18.02 of DST-Fusion of 40 subclassifiers. If the NoFusion classifier is used to assign labels to an example, in average it has to assign about 8 labels to be able to get all the correct labels of that example. Note that the majority of examples in this simplified EUROVOC database have 3 to 4 labels with the label cardinality 3.6. That is, half of the labels that NoFusion classifier assigns to that example are incorrect. For DST-Fusion of 40 subclassifiers, the coverage of 18.02

Table 6.1: Performance of NoFusion and DST-Fusion for different numbers of subclassifiers.

# Sub classifiers	Average Precision	Coverage	Hamming Loss	One Error	Ranking Loss
1 NoFusion	**0.75** ±0.01	**6.89** ±0.04	**0.083** ±0.002	**0.20** ±0.02	**0.078** ±0.002
5	0.70 ±0.01	7.72 ±0.03	0.095 ±0.002	0.23 ±0.02	0.095 ±0.003
10	0.68 ±0.01	8.28 ±0.09	0.102 ±0.002	0.26 ±0.02	0.106 ±0.003
20	0.64 ±0.01	8.93 ±0.08	0.106 ±0.002	0.30 ±0.03	0.120 ±0.004
40	0.51 ±0.01	18.02 ±0.75	0.111 ±0.001	0.40 ±0.03	0.349 ±0.019

out of the total number of class labels 30 means the classifier is hardly getting labels assigned correctly. We hypothesize that this failure of DST-Fusion in this case may be due to the fact that 125 features are not enough to enable induction of sufficiently good subclassifiers.

On the upside, Figure 6.6 indicates that, for the performance loss, we are compensated by substantial savings in the CPU time. For 5 subclassifiers, the CPU time consumed by induction using the DST-Fusion represents about 20% of the NoFusion CPU time, and for 10 subclassifiers, only 15%, while the performance as measured by the average precision is less affected, a decrease by 5 to 7%. For practical applications, using exceedingly many subclassifiers would cause too much precision loss to be justified for the savings in time, however.



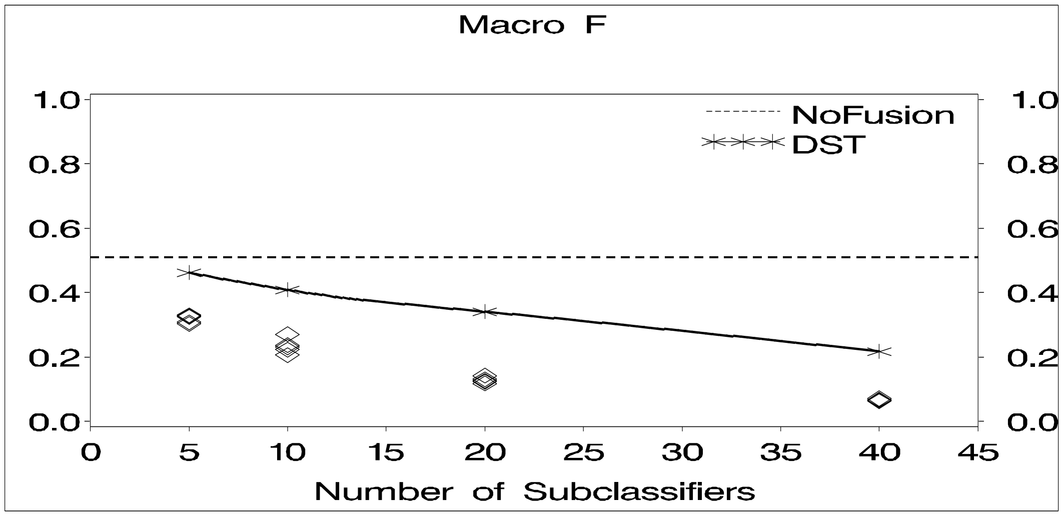


Figure 6.5: Micro-average and macro-average F_1 from 5-fold CV of DST-Fusion when varying number of subclassifiers. No-Fusion is simply DST-Fusion of one subclassifier and is included for comparison. $\diamondsuit$ indicates F_1 of each subclassifier.

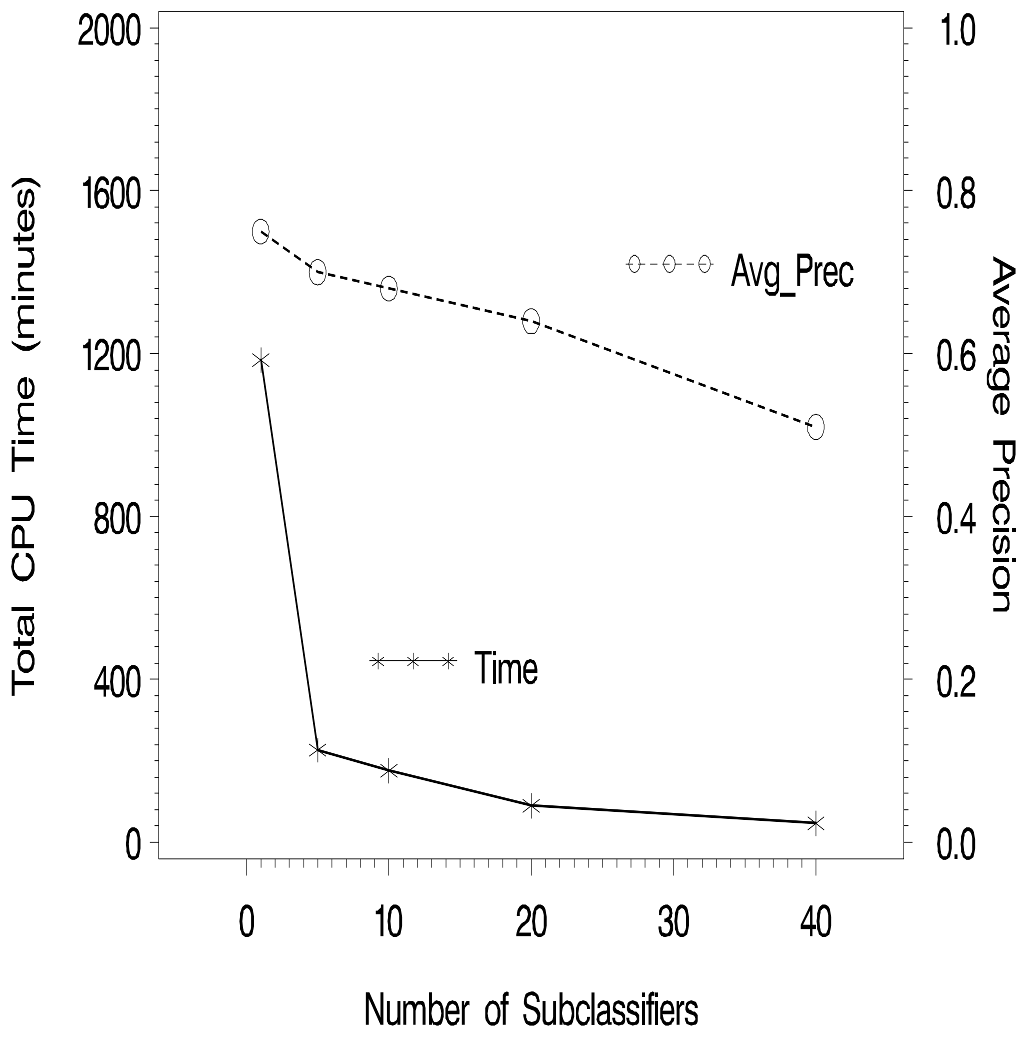


Figure 6.6: Total CPU time and average precision of DST-Fusion when varying number of subclassifiers, 5-fold CV.

6.2.3 Effect of number of boosting rounds

Since our classifier induction system requires a baseline induction algorithm to run on feature subsets and AdaBoost.MH is opted for, we want to know to what extent the quality of the baseline algorithm can be manipulated by changing the number of boosting rounds so that the final classification results from fusion can be improved. In the experiments of previous sections, the number of boosting rounds was made to depend on the number of features, i.e., we used a fixed percentage, 10%, of the number of features as the number of rounds. Here we carry out experiments with just AdaBoost.MH (or NoFusion) under the setting resembling to that in Section 6.2.1, except that the number of boosting rounds is varied as 10, 50, 100, and 250. There are 500, 1,000, 2,000, and 4,000 features divided into 5 equally-sized subsets with 20% overlapping with other subsets.

The average precision of AdaBoost.MH is shown in Figure 6.7. For a given feature set, generally increasing the number of boosting rounds will increase the average precision. But it would not be worthwhile to use too large number of rounds because after a sufficient number of iterations has been reached, all useful information from features is likely to have been utilized, and there will be no much more gain in accuracy. Furthermore, we see that when the number of boosting rounds is 10, no matter how many features we have, we get the same average precision. This tells us that 10 boosting rounds is not enough to extract necessary information from the many features we have to do a good classification. The results of this experiment demonstrate that many features in EUROVOC are not informative. In addition, it supports our decision to use the number of boosting rounds equal to 10% of the number of features in the experiments.

In the next set of experiments, we look at how increasing the number of boosting rounds of *AdaBoost.MH* can help improve DST-Fusion to compete with NoFusion.

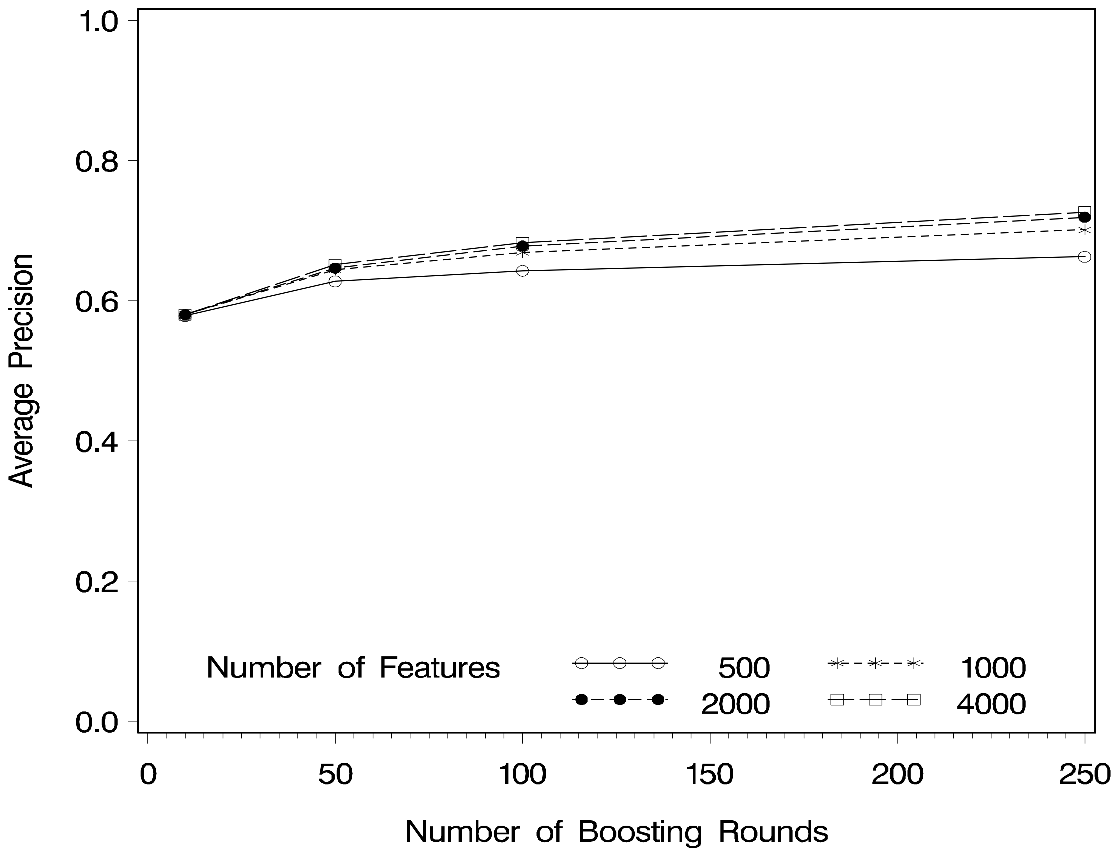


Figure 6.7: Average precision of AdaBoost.MH when using different numbers of boosting rounds.

Table 6.2: Mean and standard deviation from 5-fold CV of 5 evaluation measures for DST-Fusion of 5 subclassifiers using 1,000 features each.

# Boosting Rounds	Average Precision	Coverage	Hamming Loss	One Error	Ranking Loss
100	0.70 ±0.01	7.72 ±0.03	0.095 ±0.002	0.23 ±0.02	0.095 ±0.003
200	0.73 ±0.01	7.24 ±0.04	0.09 ±0.001	0.21 ±0.01	0.09 ±0.003
300	0.75 ±0.01	6.97 ±0.02	0.09 ±0.001	0.19 ±0.01	0.08 ±0.002
400	0.76 ±0.01	6.81 ±0.02	0.08 ±0.001	0.18 ±0.02	0.08 ±0.002

All 4,000 features are used to create 5 overlapping subsets of 1,000 features each and vary the number of boosting rounds as 100, 200, 300, and 400 for every case. It is clear from the results summarized in Table 6.2 that the performance of DST-Fusion systematically rises, but at a slow pace, with the increasing number of rounds.

This said, one has to make sure that the induction does not incur impractically large costs. Figure 6.8 indicates that the computational costs grow linearly in the number of boosting rounds, in an exchange for not-so-impressive performance gain. The average precision and the total CPU time of NoFusion (AdaBoost.MH run on 4,000 features set using 400 boosting rounds) are added in the figure as a reference. It is interesting to observe that DST-Fusion can perform comparably to or even better than NoFusion if the number of boosting rounds for inducing subclassifiers is large enough. The micro- and macro-average F_1 of DST-Fusion increase and even outstrip those of NoFusion when the number of boosting rounds is at least 200. At that point the CPU time required to run DST-Fusion is still only half of that required by NoFusion. Concerning average precision, DST-Fusion reaches the same accuracy as

NoFusion with 300 boosting rounds. More than 300 rounds would make DST-Fusion better than NoFusion, but that will call for longer total CPU time which may not be desirable.

Another point of interest regarding the number of boosting rounds is how DST-Fusion performs in comparison to NoFusion if both methods use the same number of boosting rounds irrespective of the number of features. Figure 6.8 illustrates one such situation where DST-Fusion uses 400 boosting rounds on each feature subset and it is found that DST-Fusion is better than NoFusion that also uses 400 boosting rounds on every performance criterion. More experimental results for other numbers of boosting rounds are included in Appendix C. All results suggest that DST-Fusion usually outperforms NoFusion when both of them use the same number of boosting rounds. However it becomes obvious that DST-Fusion loses its computational advantage over NoFusion. The total processing time of DST-Fusion appears to be close to or higher than that is required by NoFusion. This may not be necessarily a reason to discourage the use of DST-Fusion. If we are not subject to time constraint, but are restrained only by computer capacity, we could as well consider using the DST-Fusion in place of NoFusion in order to gain greater accuracy.

6.2.4 Classifying EUROVOC subtree data

EUROVOC database has hierarchically organized category structure. In previous sections, experiments were carried out on the simplified database that contains 10,000 documents described by 4,000 features and classified into 30 top-level classes of the classification hierarchy. In this section we want to know the behavior of DST-Fusion when dealing with classes in the lower-level of hierarchy. A top-level class of the simplified database is selected for study. Its subtree has 3,197 documents classified into

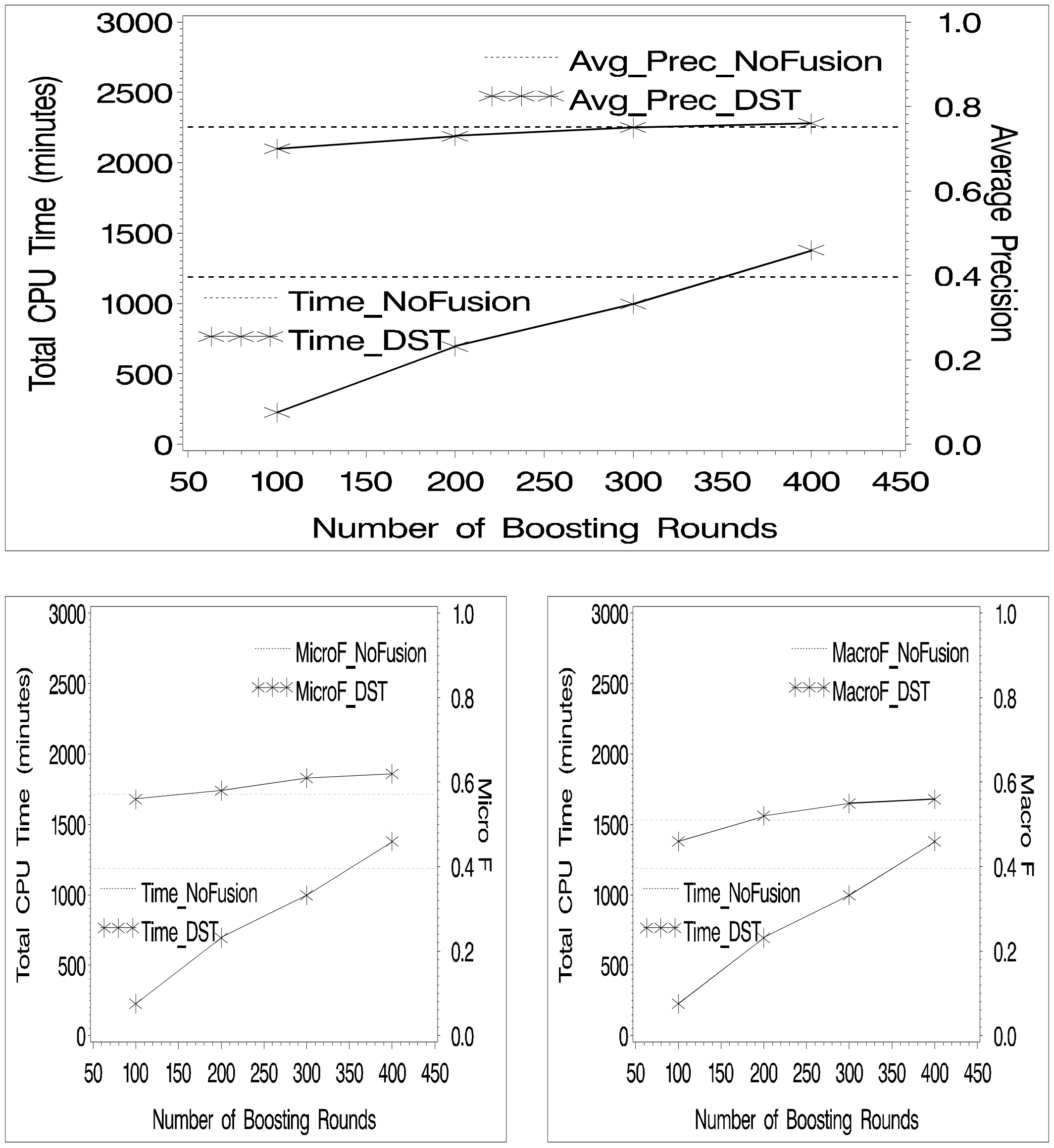


Figure 6.8: Total CPU time, average precision, micro-average and macro-average F_1 of DST-Fusion when varying number of boosting rounds, 5-fold CV. Total CPU time and corresponding performance measures of NoFusion are shown as reference.

7 classes. Out of 4,000 features, 109 features have common or constant values across all classes, which means they do not have any power to discriminate classes. The distribution of classes in this subtree is shown in Figure 6.9. These data are imbalanced; there is one class that has few observations compared to others. Figure 6.10 shows the number of documents that have different numbers of labels. The majority, 60%, of the subtree data are single-labeled, and the rest, 40%, are multi-labeled with the number of labels ranging from 2 to 4. The subtree has label cardinality 1.49 and label density 0.21.

We apply DST-Fusion on 5 overlapping feature subsets of equal size, 125, 250, 500, and 1,000 features. The results from DST-Fusion and NoFusion on the subtree data are in Figure 6.11. We reach similar conclusions as in the experiments of top-level classes. DST-Fusion cannot perform as well as NoFusion, but the difference in the performance is fairly small and it does not seem to depend on the number of features. DST-Fusion is still more accurate than individual subclassifiers. Figure 6.12 shows the CPU time required to run DST-Fusion and NoFusion. Because there are much less number of classes (7 versus 30 classes) and less number of documents (3,197 versus 10,000 documents) involved in subtree, the CPU time for both methods is many times less than what needed to process the whole simplified database. Again we see that DST-Fusion can save us processing time, although the savings may not be as great as we have seen from the top-level classification. For 500 features, DST-Fusion cuts down half of the time spent by NoFusion and for 4,000 features, DST-Fusion uses less than 1/3 of the time needed for NoFusion. Hence from these experiments, we have verified that DST-Fusion has the ability to handle subtree classes.

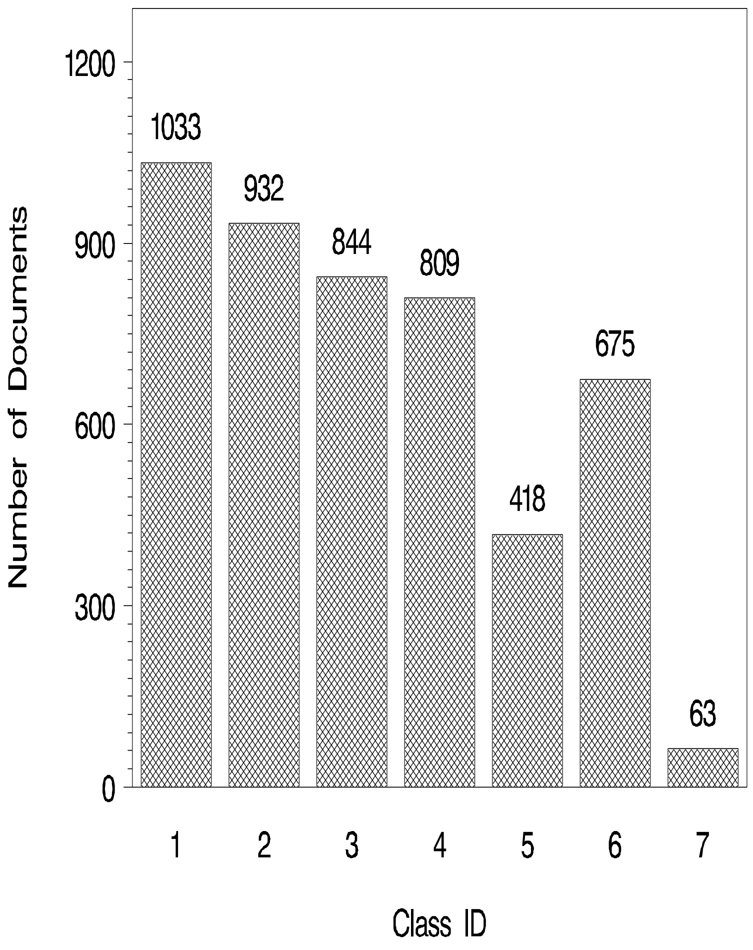


Figure 6.9: Class distribution of a EUROVOC subtree

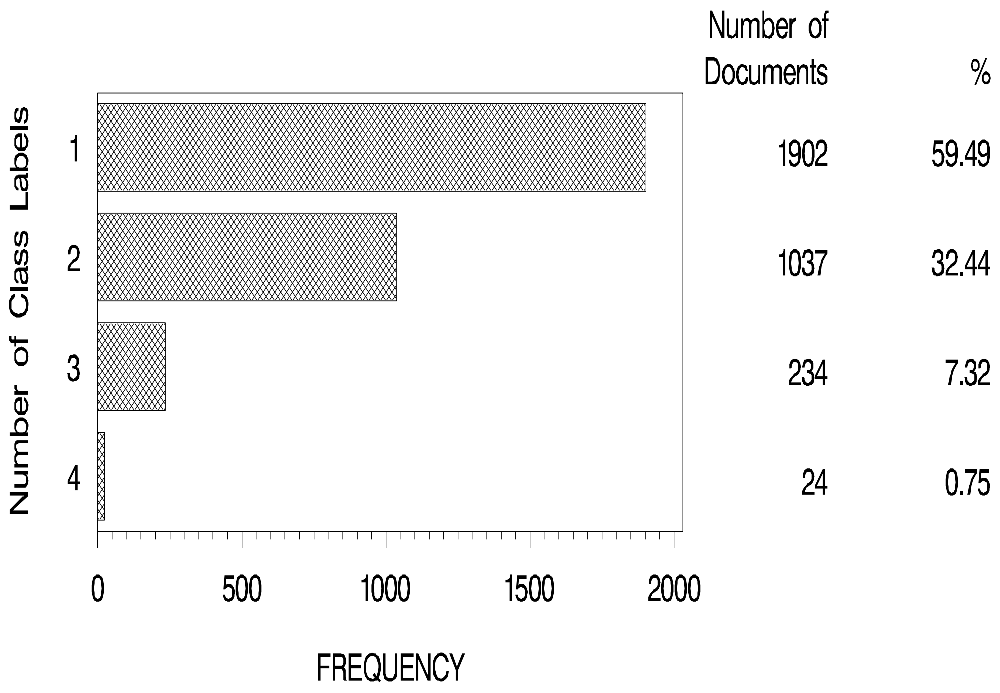


Figure 6.10: Number of documents in a EUROVOC subtree having different number of class labels.

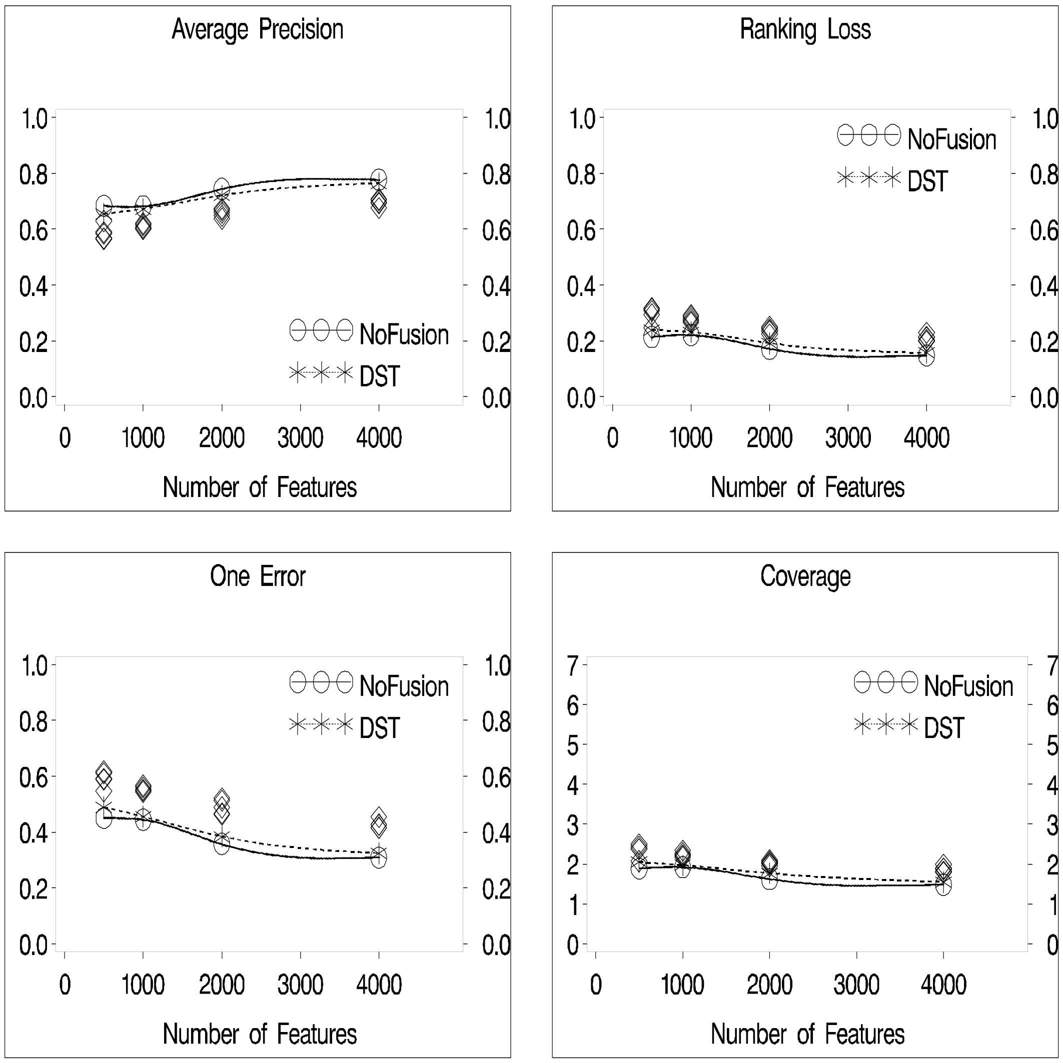


Figure 6.11: Performance of DST-Fusion and NoFusion on EUROVOC subtree data when varying number of features. ◇ indicates measurement of each subclassifier.

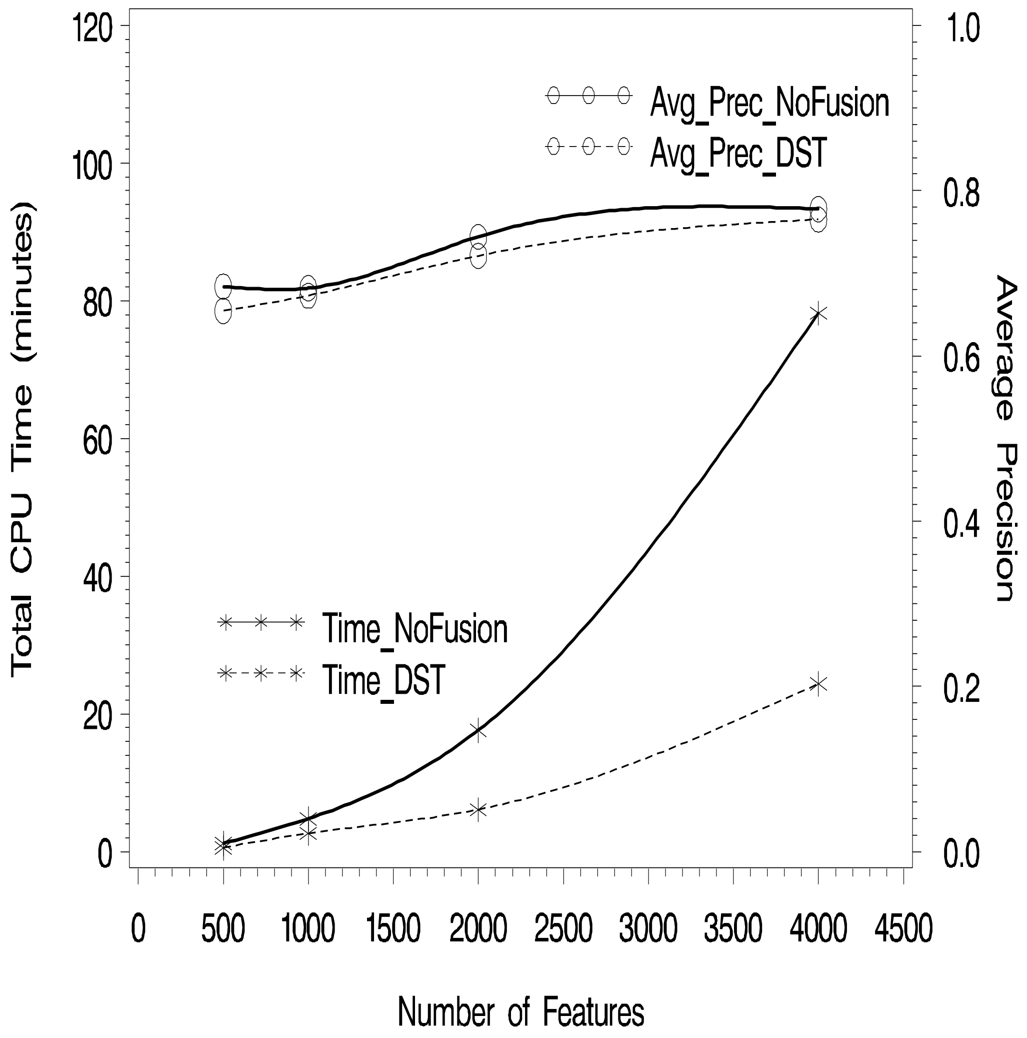


Figure 6.12: CPU time and average precision of DST-Fusion and NoFusion on EUROVOC subtree data when varying number of features.

6.3 Experiments with RCV1-v2 Data

Having seen promising results with EUROVOC data, we are interested to see how the DST-Fusion performs on other data sets. Here we select a data set from Reuters collection, Reuters Corpus Volume 1 (RCV1), to experiment with. Reuters, Ltd. is the largest international text and television news agency whose editorial division produces more than 11,000 stories a day in 23 languages. RCV1 is a high quality and large corpus of newswire stories drawn from one of Reuters' online databases that consist of only English language stories, and was released and made available by Reuters in 2000 for research purposes (Rose et al. 2002). It contains a corpus of over 800,000 manually categorized Reuters news articles from 20 Aug 1996 to 19 Aug 1997. A modified version of this corpus, called the Reuters Corpus Volume 1 version 2 (RCV1-v2), was extensively documented by Lewis et al. (2004) after they corrected various inconsistencies in the original corpus. RCV1-v2 is a standardized collection of documents suitable for analysis and testing; it provides benchmark data for testing several widely used supervised learning methods.

Every data set has 3,000 documents, each being labeled by one or more topics in the set of 101 topic categories. Data are very sparse; there is a total of 47,236 features, many of which have zero values. We randomly select only 2,000 features from a little over 2,000 features having non-zero values in more than 18 documents. Shown in Figure 6.13 are the class distributions of documents in one training set and its corresponding test set. The two distributions are in agreement indicating both data sets are from the same population. The data are imbalanced; two major classes have more than 500 documents, and the rest of classes have less than 500 documents with many classes under 100 documents. Figure 6.14 displays the multi-label nature of documents in these two data sets. The majority, 60%, of documents have 2 to 3 labels, 15% have a singe label, and the remaining have more than 3 labels. The highest number of labels a document has is 11 in the training set, and 13 in the test set. For the other 4 training sets and 4 test sets, their distributions and the multi-label patterns look identical to the two sets presented here. See Appendix D for a complete profile of all data. The average label cardinality from all data sets is 2.9, and the label density is 0.03.

We vary the number of features in the experiments as 500, 1,000, 1,500, and 2,000, and for each case, the feature set is divided into five equally-sized subsets with 20% of the features in each subset overlapping with other subsets. Since features in RCV1-v2 data are presumably more informative than features found in EUROVOC data, we set the number of boosting rounds used by AdaBoost.MH to 20% of the number of features. The results in Figures 6.15 and 6.16 are the standard evaluation criteria of DST-Fusion and NoFusion. DST-Fusion performs better than each of subclassifiers that it combines. It has higher micro-average recall than NoFusion, but its micro-average precision is always lower. In consequence, micro-average F_1 of DST-Fusion is slightly smaller than that of NoFusion. For macro-averaging measures, DST-Fusion has higher macro-average recall with more than 1,000 features and same macro-average F_1 as NoFusion with 2,000 features. The multi-label performance measures in Figure 6.17 all indicate that DST-Fusion is a little less effective than NoFusion.

On the efficiency side, Figure 6.18 exhibits the superiority of DST-Fusion; it is roughly three times faster than NoFusion. While both DST-Fusion and NoFusion perform better, though marginally, with more features, the difference in the performance between the two is smaller as more features being used. This means that in an actual application with a large number of features we can use DST-Fusion instead of NoFusion without too much accuracy loss while reducing the processing time down to one-third.

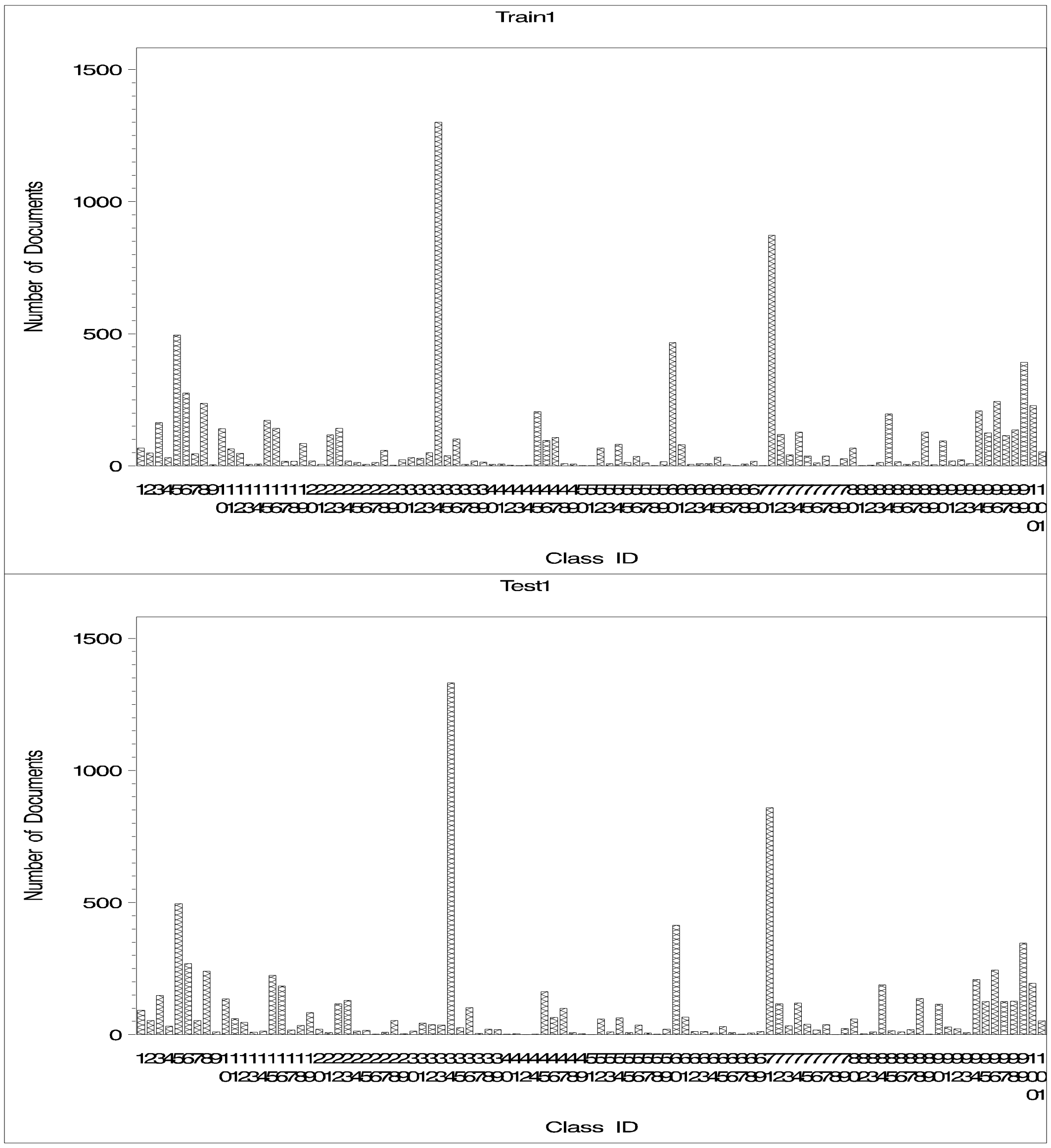


Figure 6.13: Class distributions of documents in RCV1-v2 experimental training data and test data, non-disjoint classes. 3,000 documents in each set.

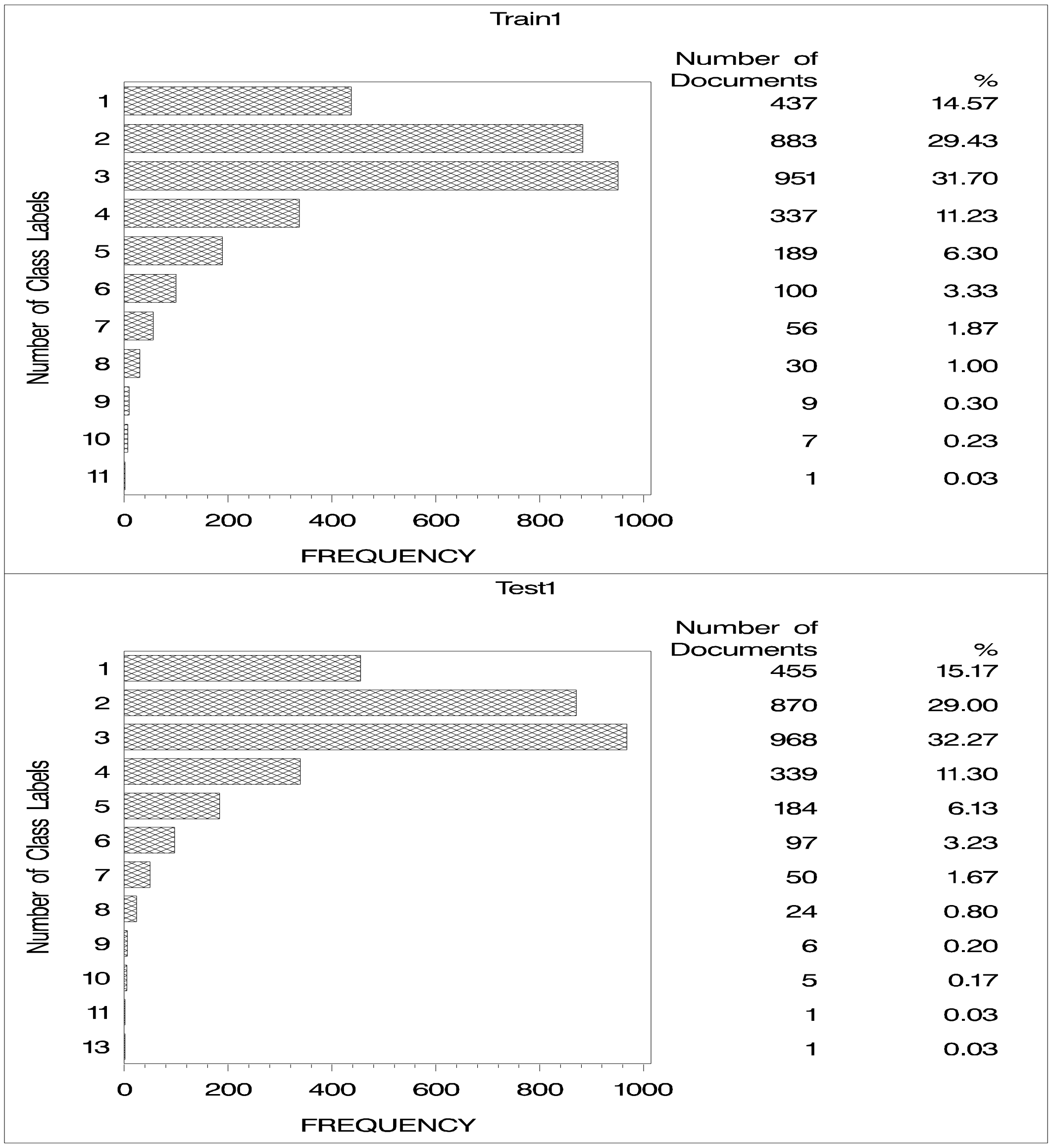


Figure 6.14: Number of documents in RCV1-v2 experimental training data and test data having different numbers of class labels.

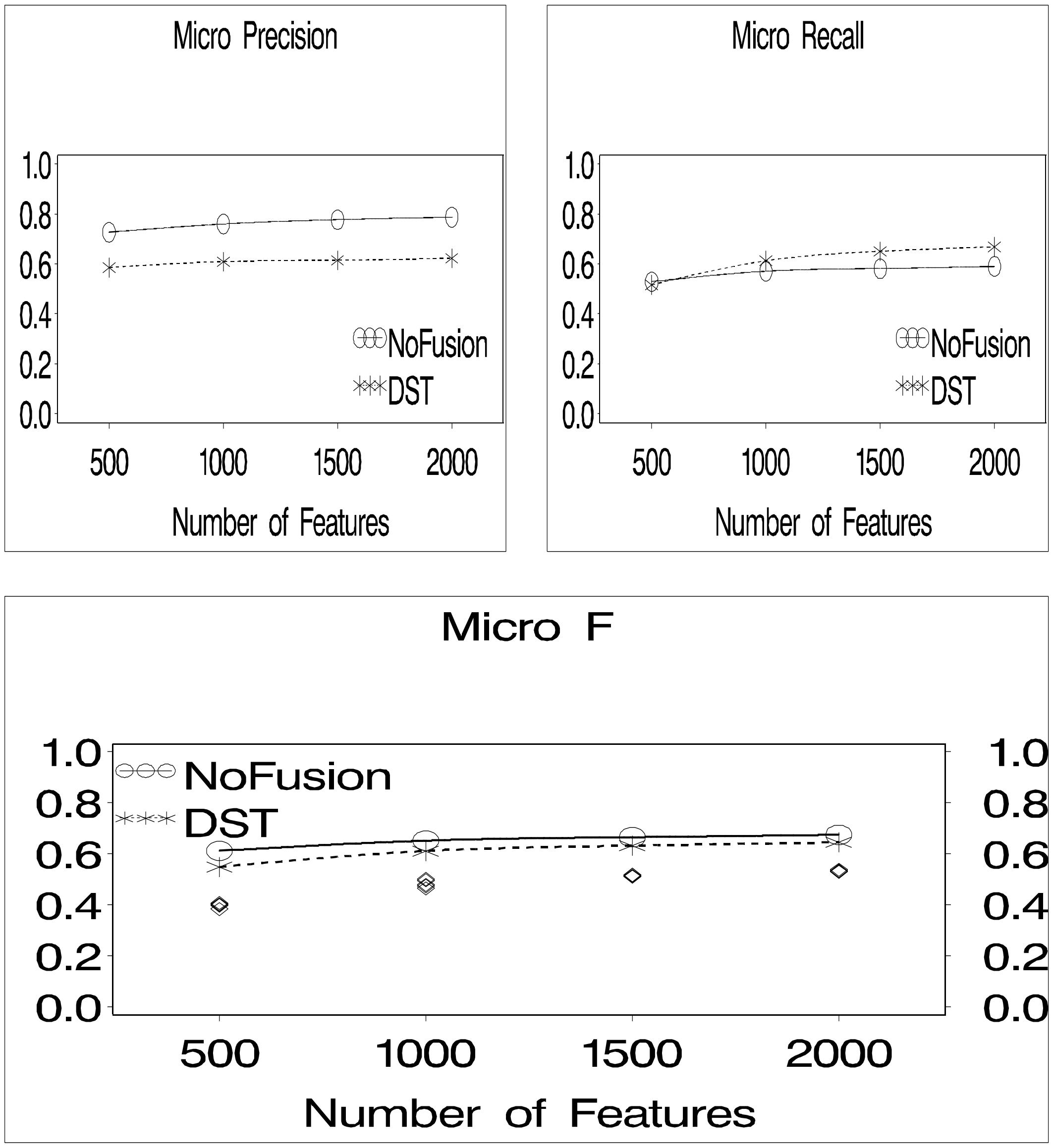


Figure 6.15: Micro-average recall, precision, and F_1 of DST-Fusion and NoFusion for RCV1-v2 data.

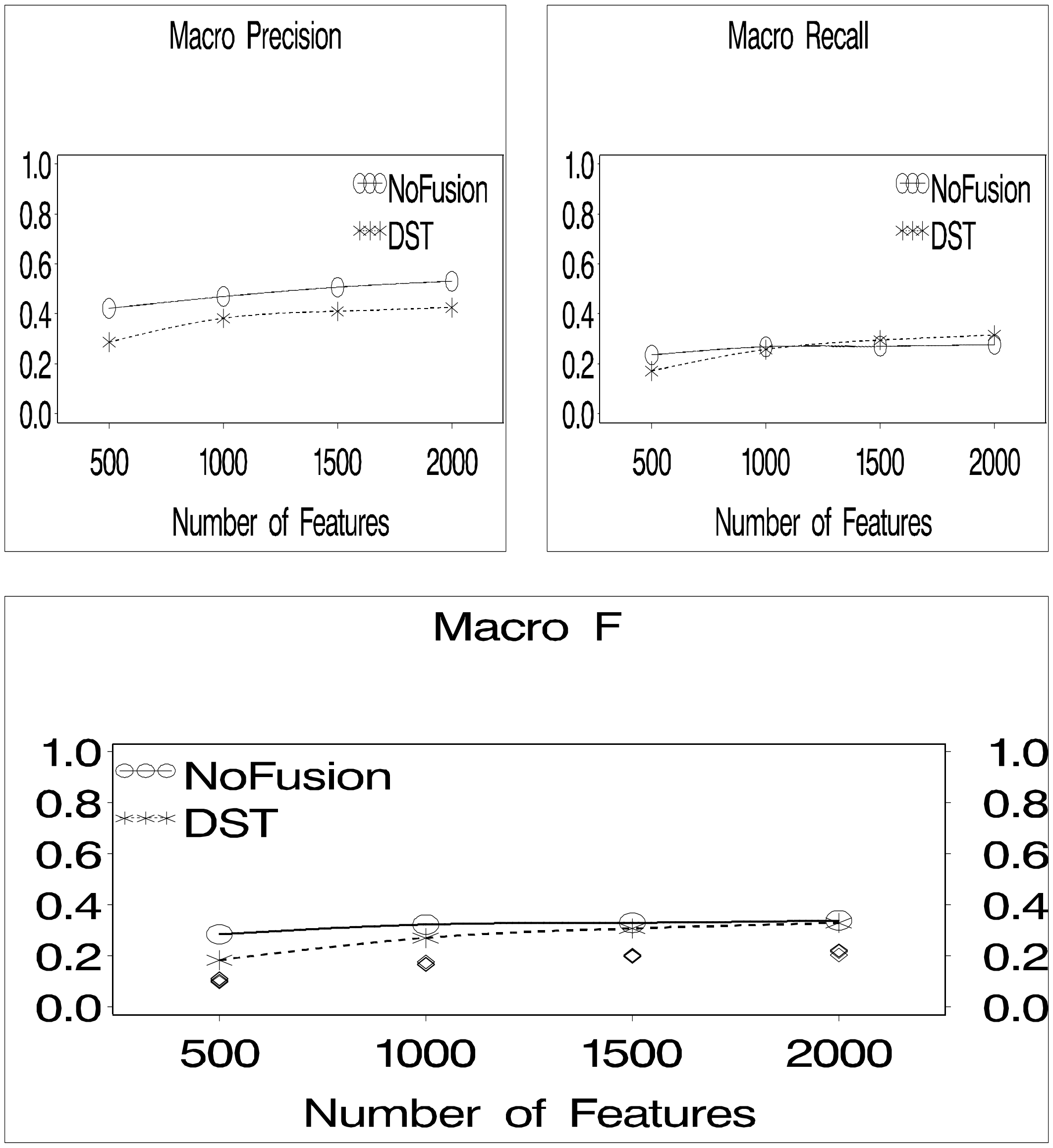


Figure 6.16: Macro-average recall, precision, and F_1 of DST-Fusion and NoFusion for RCV1-v2 data.

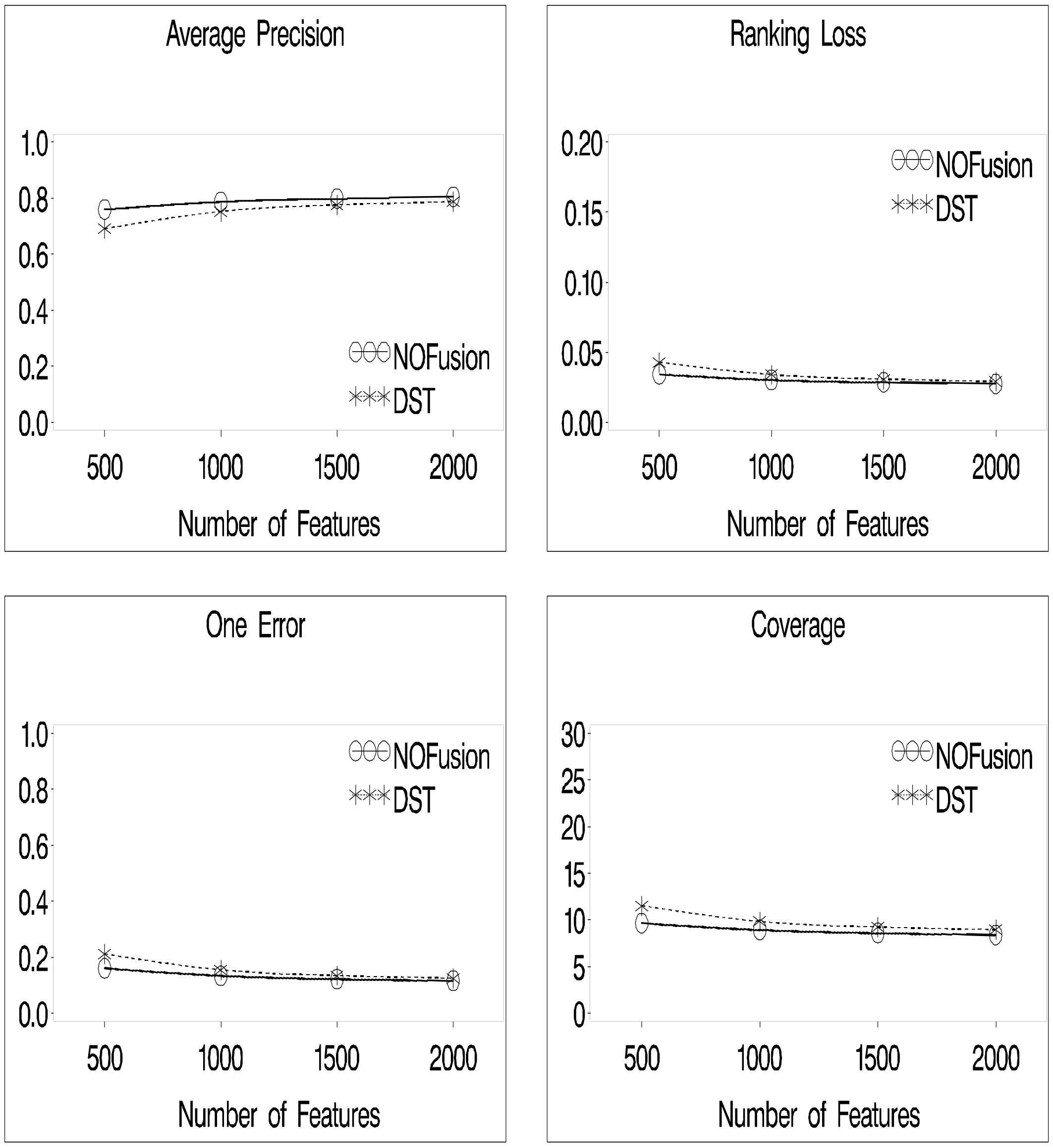


Figure 6.17: Performance of DST-Fusion and NoFusion for RCV1-v2 data as measured by multi-label evaluation criteria.

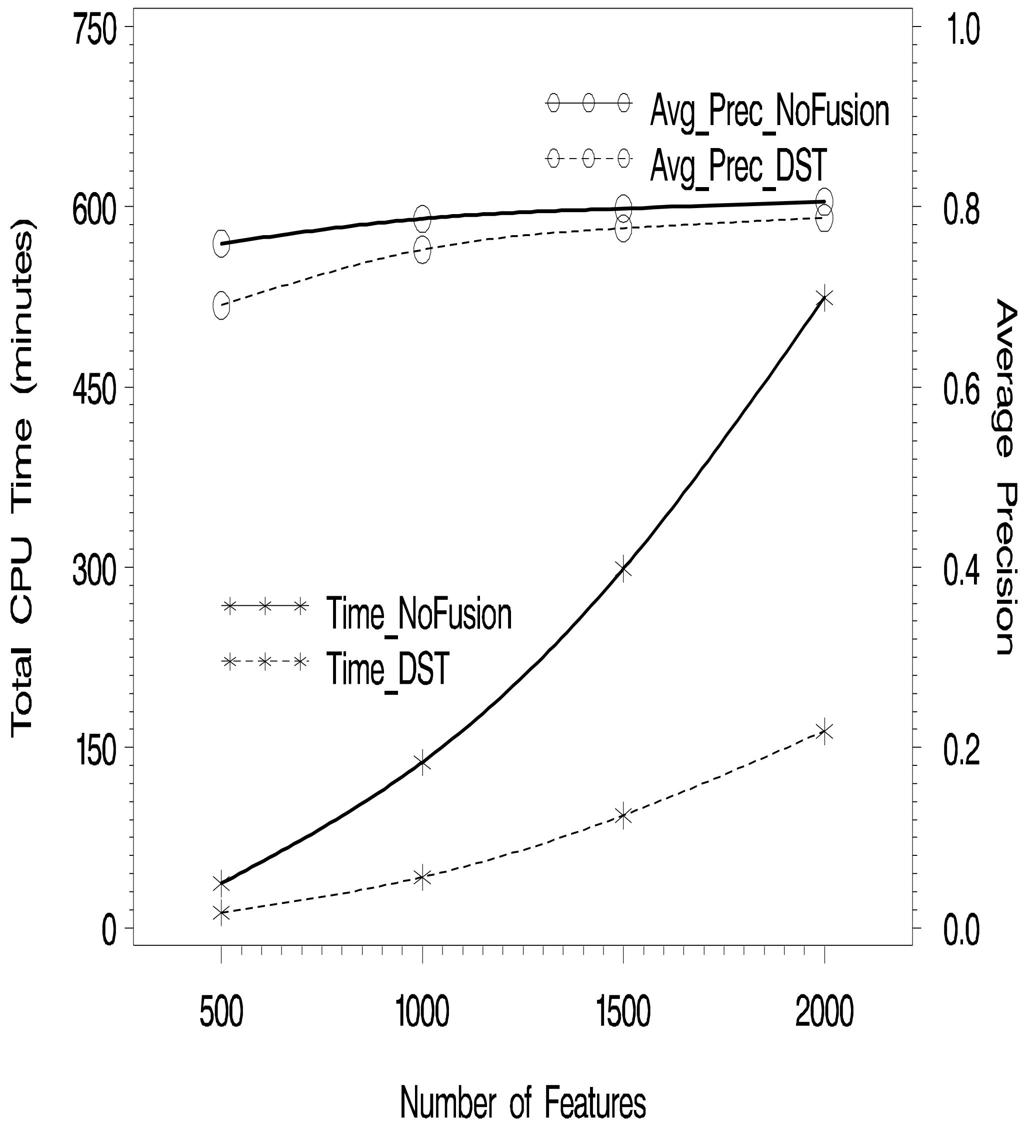


Figure 6.18: Total CPU time and average precision of DST-Fusion and NoFusion for RCV1-v2 data.

6.4 Benefits of Fusion

Previous sections have provided experimental evidence regarding the performance of DST-Fusion. Results from experiments on two data sets agree. In summary, DST-Fusion generally suffers some accuracy loss when compared to the classifier induced from all features processed collectively, and effectiveness of DST-Fusion decreases with increasing number of subclassifiers. This disadvantage, however, is well compensated by the significant gain in induction speed. When we subdivide the feature set into a few subsets, we have seen that DST-Fusion brings about convincing computational savings without too much compromising classification performance. The savings in time is larger when having more features while the precision loss is about the same regardless of the number of features. Additionally, the experiments illustrated that the performance of DST-Fusion can be boosted to reach the same level of accuracy as NoFusion by increasing the number of boosting rounds of AdaBoost.MH that we use as a baseline induction algorithm.

Classifier learning is the most time and resource consuming task for text categorization. Even if time permits, for a very large training data set, a fast CPU and big memory are needed to train data, which could be an obstruction for usual induction. Our proposed fusion approach can alleviate the problem of computational costs associated with the current no-fusion approach. DST-Fusion can scale up well with respect to high numbers of features. It yields a speedup for induction methods where the computational complexity is superlinear in the number of features. Indeed the CPU time in Figures 6.4 and 6.18 shows that DST-Fusion exhibits such a speedup approximately 3 times faster for RCV1-v2 data and EUROVOC data with the number of features up to 2,000 and at least 5 times faster for EUROVOC data with 4,000 features.

DST-Fusion works by combining evidence from subclassifiers that are generated independently from feature subsets. This makes DST-Fusion a preferred method because it is more efficient, more convenient, and faster to work with small data sets, particularly when we have limited computing resources. For example, the memory requirements of each separate run is less with fewer data. Besides, the ability to split the classification task gives us more flexibility in processing data. The independence of subclassifier induction means that the process of inducing subclassifiers needs not wait for one subclassifier to finish before another can be induced. In other words, the induction can be done simultaneously for every subclassifier. The implication is that DST-Fusion can fully take advantage of parallel processing. Figures 6.19 and 6.20 show the process real time required by DST-Fusion for EUROVOC and RCV1-v2 data when parallel processing is exploited. The learning task now can be accomplished in much less time. The implementation of DST-Fusion as a parallel learning algorithm enables fast learning that is desirable in some practical applications.

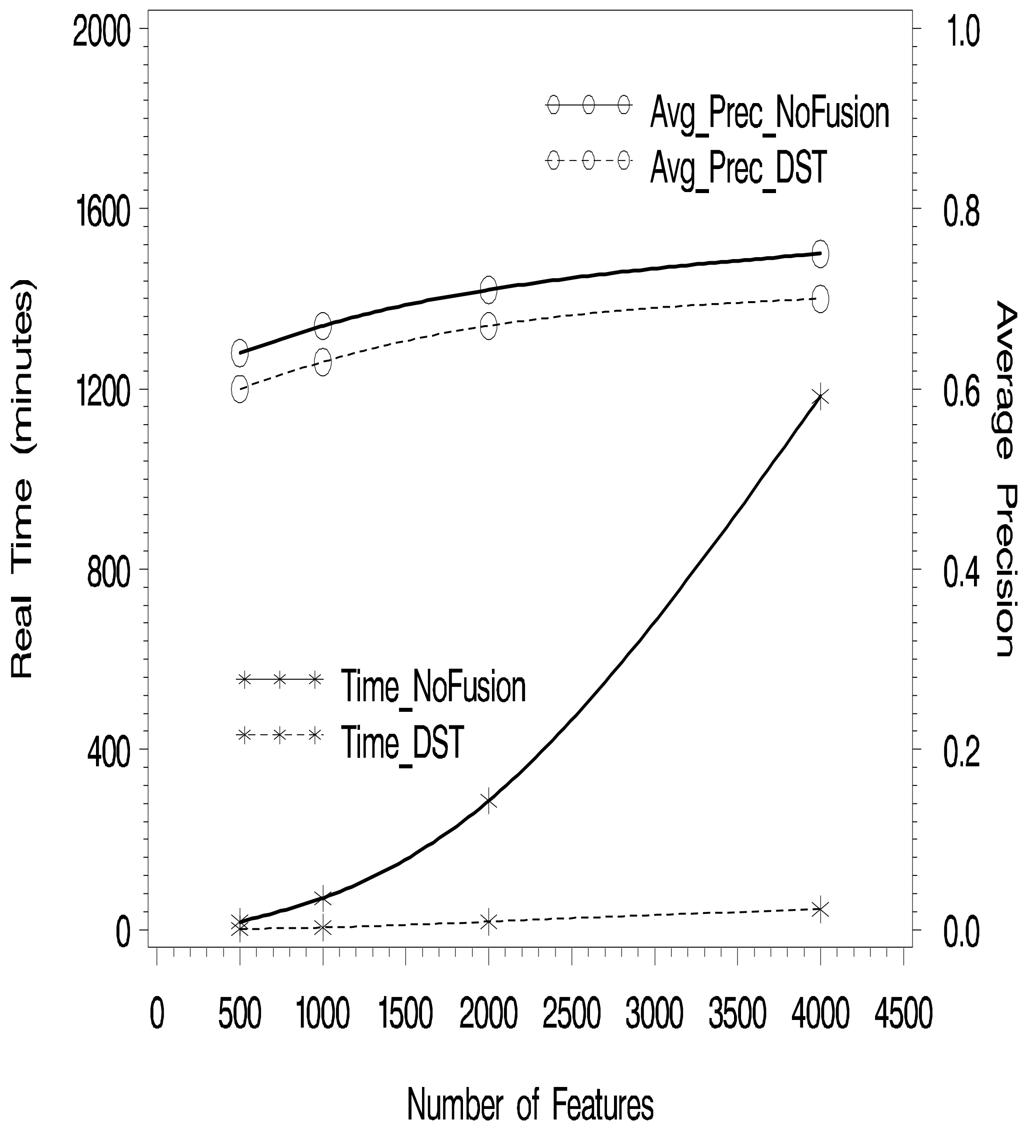


Figure 6.19: Real time and average precision of NoFusion and DST-Fusion when subclassifiers are processed in parallel, EUROVOC data

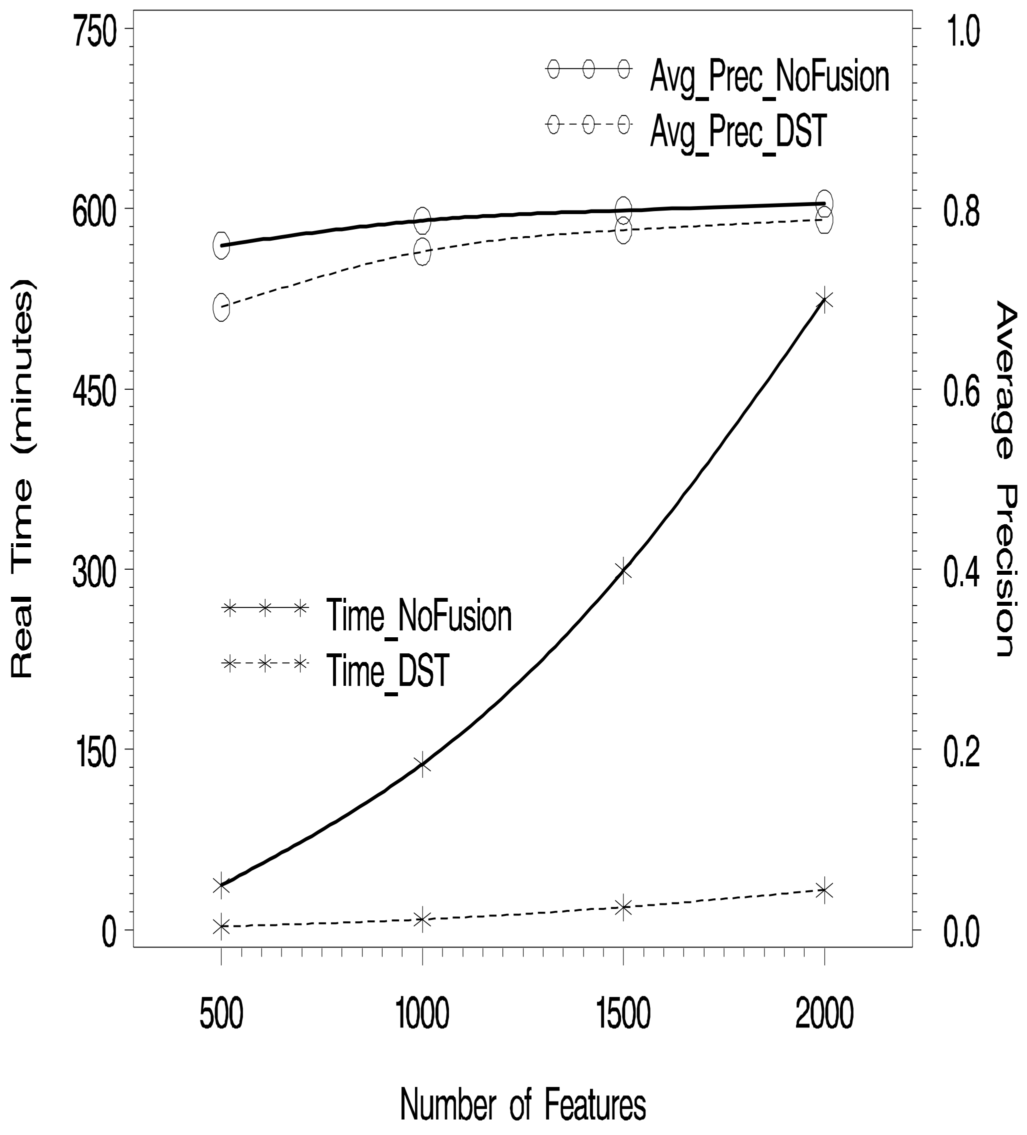


Figure 6.20: Real time and average precision of NoFusion and DST-Fusion when subclassifiers are processed in parallel, RCV1-v2 data

CHAPTER 7

Comparison of Fusion Methods

A master algorithm to combine outputs from subclassifier committees is an essential component of the proposed multi-label classifier induction system. We have developed a master algorithm called DST-Fusion based on the Dempster-Shafer theory. Dempster-Shafer's evidence combination is just one among a few classifier combination methods that we may choose to apply to the multi-label problems. Some other combination schemes that we have considered include simple or unweighted majority voting, weighted majority voting, and weighted sum methods.

These three classifier combination methods were originally created for single-label classification. In the following sections, we modify them such that they can combine predictions of multiple multi-label classifiers. We then carry on a comparative study on the performance of these combination methods and DST-Fusion using the simplified EUROVOC database.

7.1 Voting Methods and Weighted Sum for Multi-label Problems

Majority/plurality voting system is a popular classifier combination technique used in various disciplines. Many people are more accustomed to the term "majority" voting than "plurality" voting, and often call the method majority voting when, in fact, it is plurality voting. The slight difference between the two voting systems is that majority voting requires the agreement of more than half of the committees to reach a decision, whereas plurality voting selects the candidate with the most votes (Lin et al. 2003). When there are only 2 candidates to select from, plurality voting is simply the same as majority voting. Since in our multi-label classification scheme we consider one label at a time and make a binary decision about the label, we will use the more common name, majority voting, here.

The appeal of voting arises from its simplicity, generality, and effectiveness. The implementation of majority voting is by far the simplest, and in many practical applications, there is only marginal performance difference between majority voting and more advanced, complicated combination schemes, which require greater development efforts. Theoretical justifications of the majority voting and plurality decision criteria were provided by Lam and Suen (1997) and Lin et al. (2003), respectively. Lin et al. also demonstrated that the combination of independent classifiers by voting results in dramatic accuracy improvement. This is because the examples that are incorrectly classified by one classifier have a good chance to be correctly classified by a majority of the other classifiers.

We are interested in studying the prediction effectiveness of voting methods when applied on the outputs of different multi-label subclassifiers induced by the baseline algorithm, AdaBoost.MH. AdaBoost.MH generates a set of binary classification rules, one rule for each class label with outputs yes/no for the label. We combine binary outputs from subclassifiers by operating the combiner on one label each time. Suppose the multiple classifier system comprises N subclassifiers. Each subclassifier makes its decision about a label c independently of other subclassifiers. Let $f_j(x,.)$ be the ranking function of an example x from subclassifier j, $j = 1, 2, \ldots, N$. The three voting methods considered are:

1. Simple or unweighted majority voting. Each subclassifier has one vote. Prior knowledge of the behavior of the individual subclassifiers is not assumed for this voting method. That is, the method treats each subclassifier as having the same probability of voting correctly, or, in other words, all votes are equally accurate. Therefore the majority vote rule for class label c is

   Assign the label c to example x if

   $$\sum_{j=1}^{N} \| \, f_j(x,c) \geq 0 \, \| \quad > \quad N/2$$

   The summation simply counts the votes received for class c from the individual subclassifiers. Class c is the consensus (majority) decision from the subclassifier pool if the class c receives more than half of the total votes.

2. Weighted majority voting. Not all subclassifiers are built to be good at categorizing every label. The accuracy of the subclassifier j in making a decision concerning the class c is the same as what has been defined in Equation (5.9):

   $$Acc_j(c) = P_j(c|c)P(c) + P_j(\bar{c}|\bar{c})P(\bar{c}) \tag{7.1}$$

   where $P_j(c|c')$ is the conditional probability that the subclassifier j assigns class label c to a class c' example, and $P(c)$ is the prior probability of class c.

   When combining outputs related to a class c, we weight the vote of each subclassifier by the accuracy of the subclassifier in categorizing the class c. Votes from more accurate subclassifiers receive higher weight. The weighted votes for class c and class "not" c from all subclassifiers are computed. The weighted majority vote rule for class label c is

Assign the label c to example x if

$$\sum_{j=1}^{N} Acc_j(c) \parallel f_j(x,c) \geq 0 \parallel \quad > \quad \sum_{j=1}^{N} Acc_j(c) \parallel f_j(x,c) < 0 \parallel$$

That is, if the class c is supported by subclassifiers that are more accurate, then it is more likely that the example belongs to class c.

3. Weighted sum. This variant of voting system works on the confidence scores of the binary decision rather than the binary decision itself. The confidence scores of all possible labels are weighted by the accuracies of the individual subclassifiers, and then are summed over all subclassifiers to give the overall confidence in the labels. The sum of weighted confidence score is used to determine if the label should be assigned to the example.

   Since subclassifiers are induced from different sets of data, the confidence scores that come from the ranking functions are not truly conforming between subclassifiers and must be transformed and normalized, as explained in Section 5.3.1, before computing weighted sum. Let ${f_j}^*(x,c)$ be the normalized confidence score of class label c for the example x from subclassifier j. The weighted sums of class c and class $\bar{c}$ are $\sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,c)$ and $\sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,\bar{c})$, respectively, where $Acc_j(c)$ is the accuracy in predicting class c of subclassifier j in Equation (7.1). The label of class that has higher weighted sum value will be assigned to the example. Consider the case where we will assign class c to the example. We should have

$$\begin{aligned}\sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,c) &> \sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,\bar{c}) \\ &= \sum_{j=1}^{N} Acc_j(c)(1 - {f_j}^*(x,c)) \\ &= \sum_{j=1}^{N} Acc_j(c) - \sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,c)\end{aligned}$$

or

$$2 * \sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,c) > \sum_{j=1}^{N} Acc_j(c)$$

which leads to the following weighted sum decision rule:

Assign the label c to example x if

$$w(c) = \frac{\sum_{j=1}^{N} Acc_j(c) * {f_j}^*(x,c)}{\sum_{j=1}^{N} Acc_j(c)} > 0.5.$$

7.2 Comparing Four Fusion Methods

To gain insight on how the DST-Fusion performs as compared to the three aforementioned voting methods, we carry out experiments on the simplified EUROVOC database. We follow the same experimental setup described in Section 6.1.

Figures 7.1 and 7.2 compare the performance of the NoFusion, DST-Fusion, Weighted majority voting, and Weighted sum. For fusion methods, the feature set is split into 5 feature subsets from which 5 subclassifiers are induced and subsequently combined by each of the fusion methods. The plain majority voting is omitted from the graphs because it almost always underperforms the other approaches; the inclusion of its chart would only reduce clarity. The standard performance criteria, micro- and macro-averaging of precision, recall, and F_1, are used. The graphs show that the performances of DST-Fusion and the weighted sum are comparable to NoFusion from micro-average F_1 standpoint. A closer look reveals that DST-Fusion and weighted sum have the lowest micro-average precision but the highest micro-average recall among all methods with no exception for NoFusion, while the two majority-voting schemes have poor micro-average recall and, hence, micro-average F_1. In terms of macro-averaging, NoFusion outperforms every method. The charts also show the results of the individual subclassifiers. We see that DST-Fusion, weighted sum, and NoFusion outperform each single subclassifier, whereas the majority voting (weighted and unweighted) occasionally fails to outperform the best subclassifier.

Figure 7.3 compares the performance of the fusion methods along the multi-label performance criteria. Again, DST-Fusion and the weighted sum outperform the other fusion methods, though they still cannot compete with NoFusion on some measures. For majority voting methods, they perform poorly, and quite often they are inferior to even subclassifiers.

It is interesting to note that DST-Fusion and the weighted sum turn out to be very close in performance. Their charts are nearly undistinguishable. DST-Fusion is slightly better than weighted sum on the recall, while the weighted sum is only marginally better than DST-Fusion on the precision. That is, DST-Fusion tends to assign labels to the documents more often than the weighted sum does. When subjected to the pairwise t-test, the differences in the performance of DST-Fusion and the weighted sum are statistically insignificant. However, this does not mean that they tend to label the documents with the same classes. In fact, we have found from detailed study of classification results that each of them misclassified different documents, but they both committed about the same number of errors. We will present more in-depth analysis of this issue in the next section.

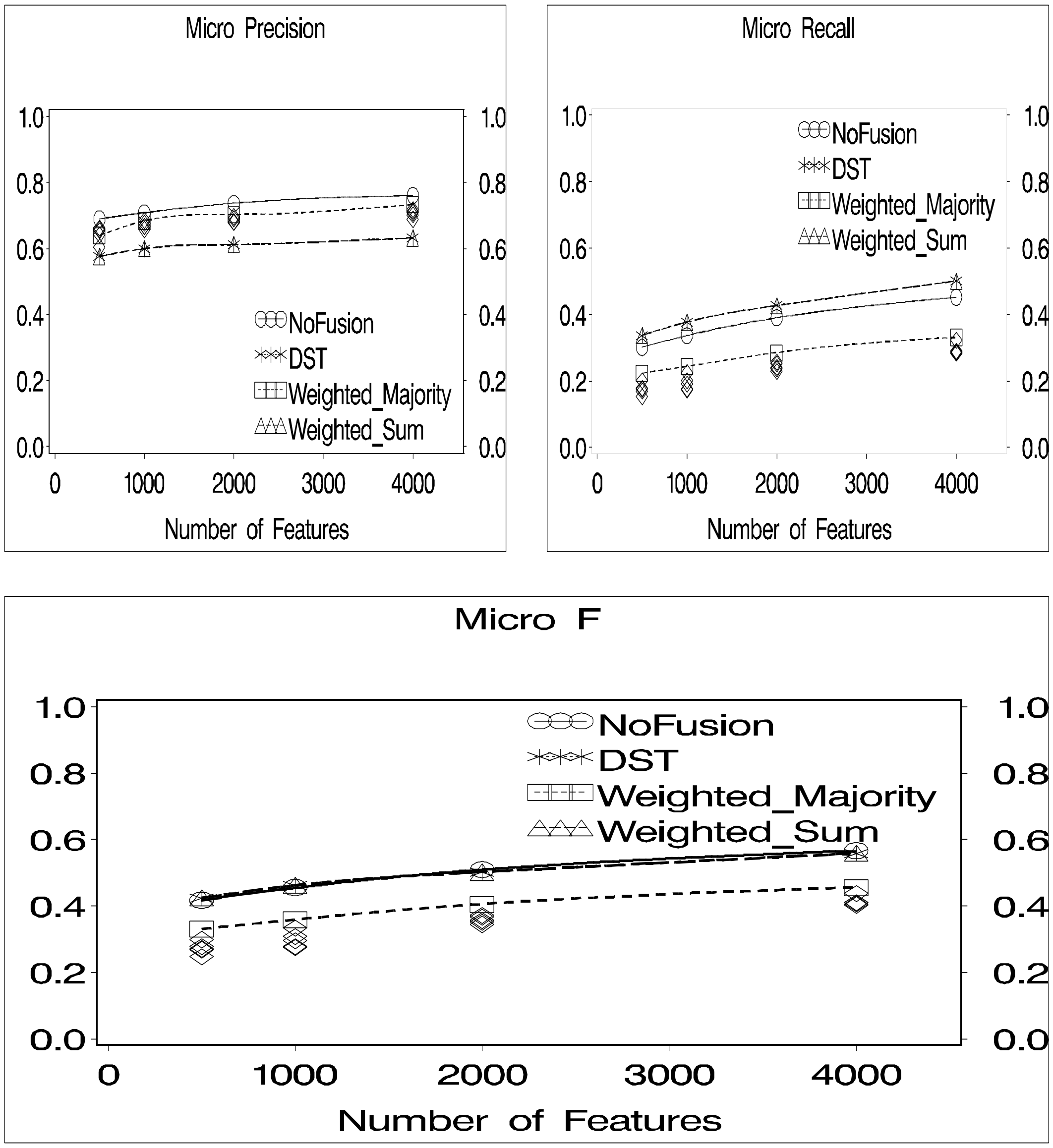


Figure 7.1: Micro-averaging of precision, recall, and F_1 for "NoFusion" and three fusion methods when varying number of features. $\diamondsuit$ indicates F_1 of each subclassifier.

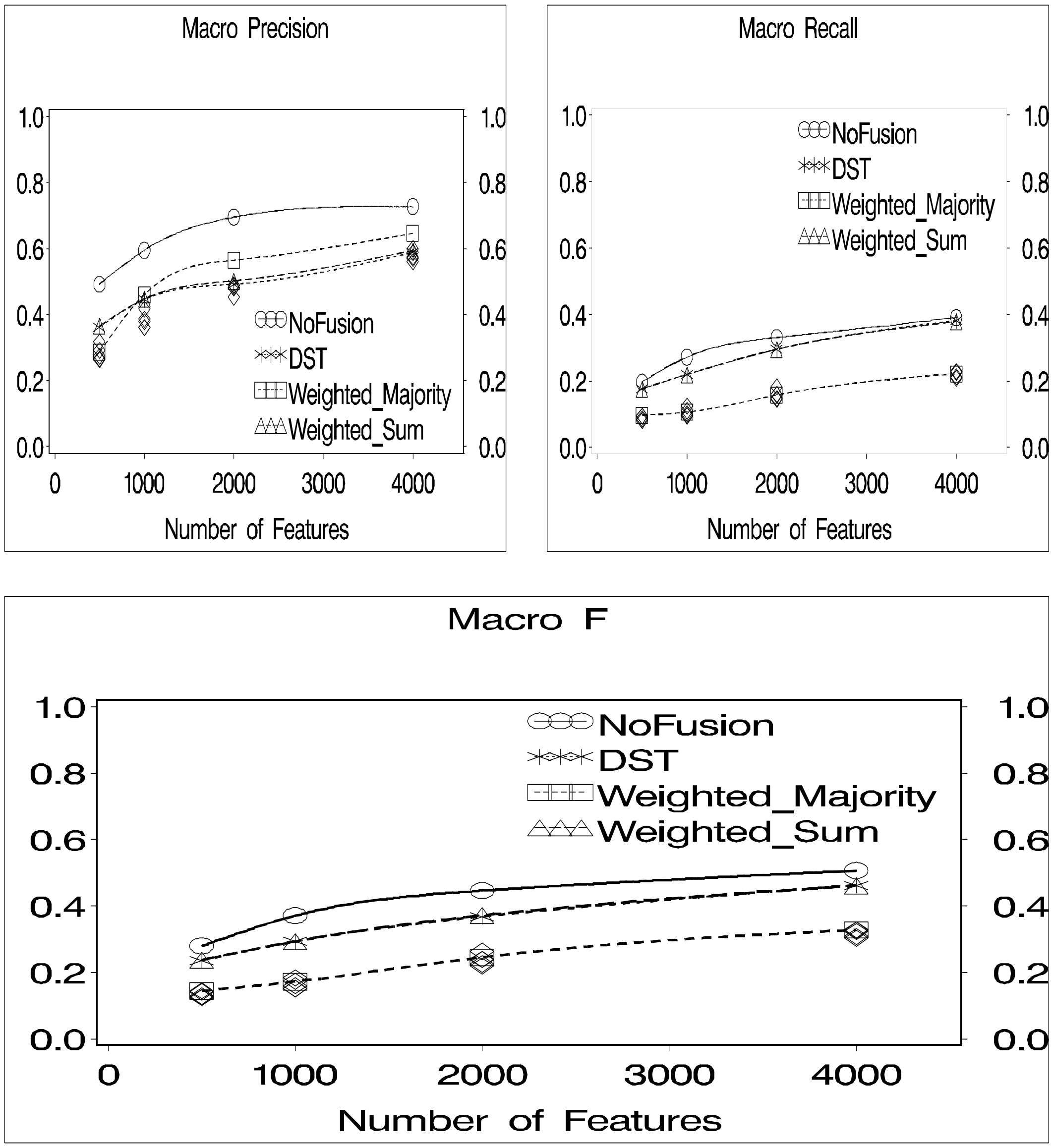


Figure 7.2: Macro-averaging of precision, recall, and F_1 for "NoFusion" and three fusion methods when varying number of features. $\diamondsuit$ indicates F_1 of each subclassifier.

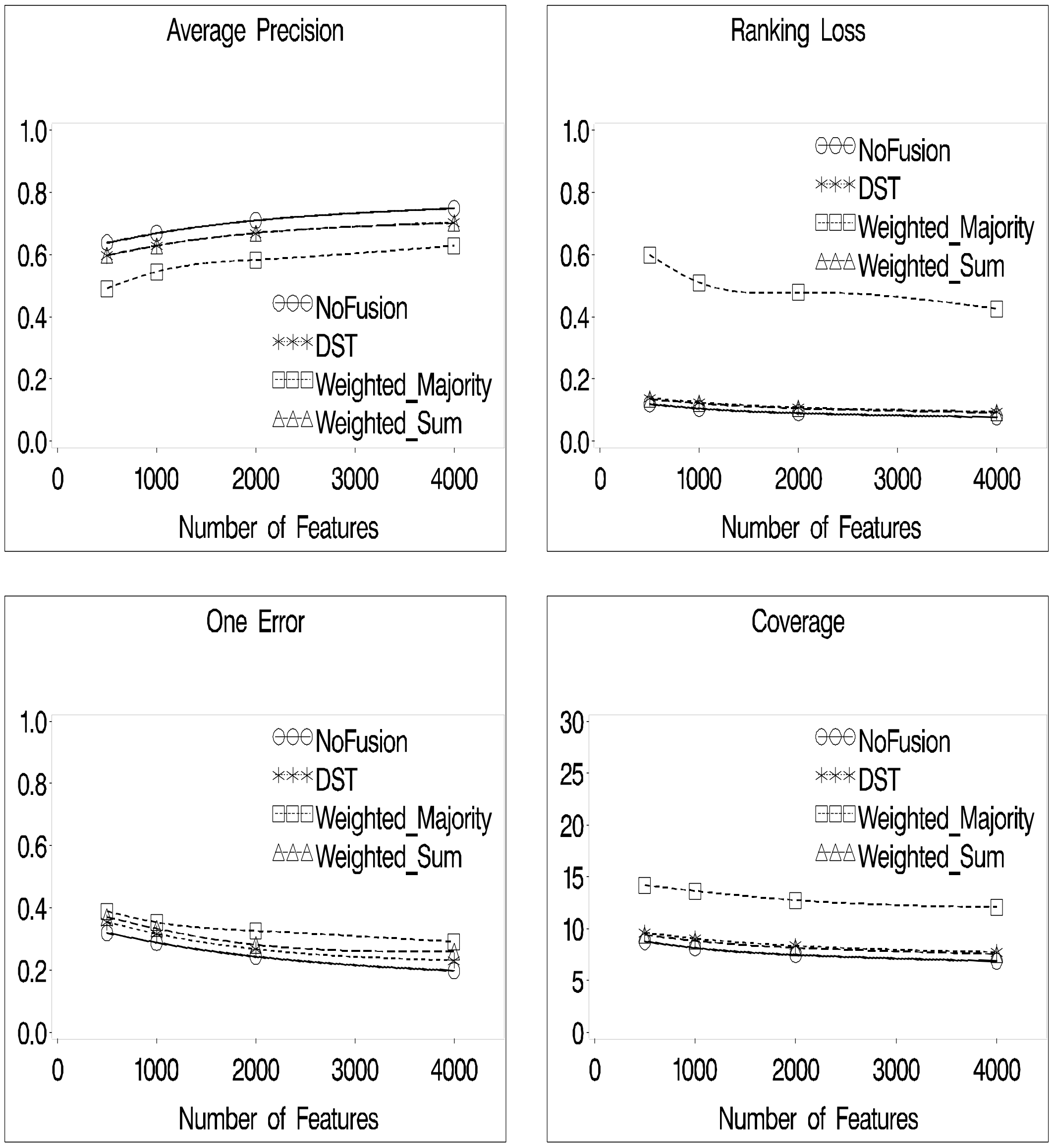


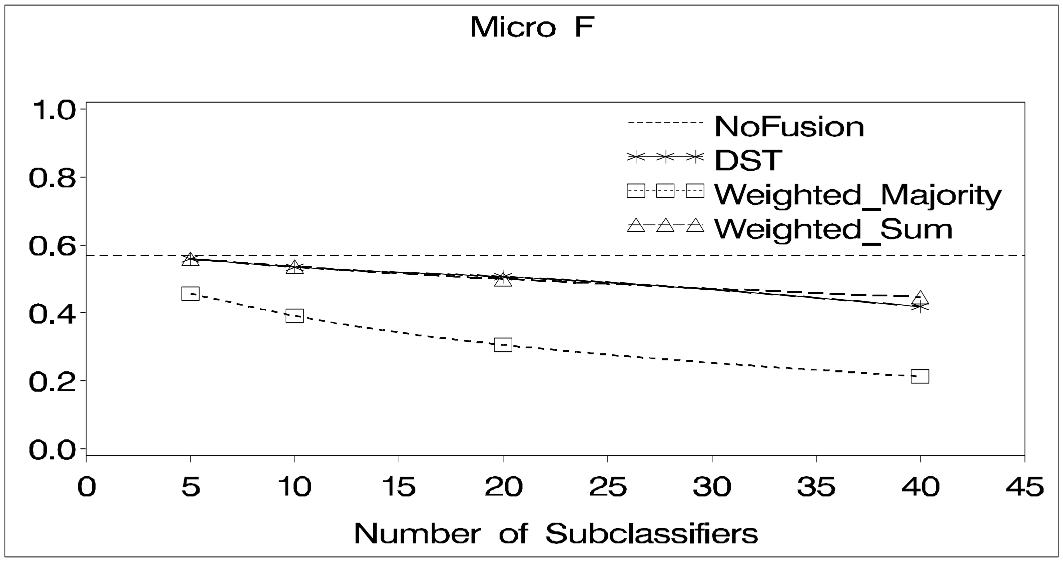
Figure 7.3: Multi-label performance measures of three fusion methods and the "No-Fusion", when varying number of features.

In the next round of experiments, we explore the performances of the fusion schemes when increasing the number of subclassifiers. Figure 7.4 shows that none of the fusion methods can hardly perform at the same level as NoFusion. The micro- and macro-average F_1 of DST-Fusion and weighted sum decay with the growing number of subclassifiers, although they still clearly outperform the other voting methods. We do not observe any discernible difference between DST-Fusion and the weighted sum, except with 40 subclassifiers that the weighted sum seems to have a little higher F_1, but it is non-significant statistically.

7.3 Detailed Analysis of DST-Fusion and Weighted Sum

Estimates of evaluation measures and statistical tests based on empirical results in the previous section showed that DST-Fusion performs at the same level as the weighted sum. Despite the test outcome, we notice that the two methods does not always output the same set of labels for the same example. Further study on case by case label prediction of test examples reveals that there are many examples where the predictions by these two methods are opposite. DST-Fusion makes incorrect decisions on some documents and the weighted sum makes incorrect decisions on some others, but at the end, the total errors are the same for both. To illustrate how DST-Fusion and weighted sum make different decisions, we will use two simple numerical examples.

Let us consider the combination of evidence for an arbitrary class, say class 1, from 2 classifiers.



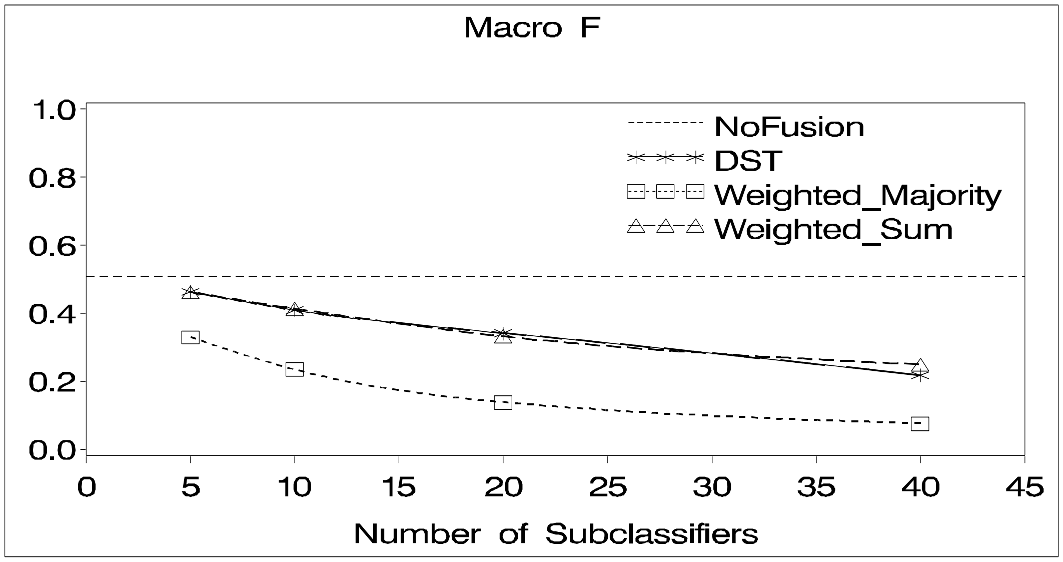


Figure 7.4: Micro- and macro-average F_1 from 5-fold CV of three fusion methods: DST-Fusion, weighted majority voting, and weighted sum, when varying number of subclassifiers. NoFusion results are shown for comparison.

1. Suppose the normalized confidence scores of class 1 for one example are 0.7 from classifier 1, and 0.05 from classifier 2. Classifiers 1 and 2 have accuracy 0.8 and 0.4, respectively. Then,

$$f_1{}^*(x,1) = 0.7, \quad f_2{}^*(x,1) = 0.05$$

$$Acc_1(1) = 0.8, \quad Acc_2(1) = 0.4$$

and, by Equations (5.10) and (5.11), we obtain the following BBAs:

$$m_1(\Theta) = 1 - Acc_1(1) = 0.2, \quad m_2(\Theta) = 1 - Acc_2(1) = 0.6$$

$$m_1(1) = f_1{}^*(x,1) * Acc_1(1) = 0.56, \quad m_2(1) = f_2{}^*(x,1) * Acc_2(1) = 0.02$$

We now apply the two fusion methods.

- Combining two classifiers by the weighted sum gives

$$w(1) = \frac{Acc_1(1) * f_1{}^*(x,1) + Acc_2(1) * f_2{}^*(x,1)}{Acc_1(1) + Acc_2(1)} \;=\; 0.4833 \;<\; 0.5.$$

Decision: Do not assign label 1 to the example.

- For DST-Fusion, we first compute K_{12} and then combine the evidence by Equation (5.13).

$$K_{12} = 1 - (m_1(1) * m_2(\bar{1}) + m_1(\bar{1}) * m_2(1)) = 0.7824$$

$$m(1) = \frac{m_1(1) * m_2(1) + m_1(1) * m_2(\Theta) + m_1(\Theta) * m_2(1)}{K_{12}} \;=\; 0.4489$$

$$m(\bar{1}) = \frac{m_1(\bar{1}) * m_2(\bar{1}) + m_1(\bar{1}) * m_2(\Theta) + m_1(\Theta) * m_2(\bar{1})}{K_{12}} \;=\; 0.3978$$

$$Bel(1) = m(1) > m(\bar{1}) = Bel(\bar{1})$$

Decision: Assign label 1 to the instance.

In this example, DST-Fusion leans toward classifier 1 and chooses class 1 because classifier 1 has higher accuracy and classifier 1 believes in class 1. On the contrary, weighted sum does not choose class 1 because classifier 2 is very confident that the example is not class 1. The weighted sum seems to ignore the fact that classifier 2 makes correct predictions less than half of the time ($Acc_2(1) = 0.4$).

2. Now suppose the normalized confidence scores of class 1 for another example are 0.7 from classifier 1, and 0.4 from classifier 2. Classifiers 1 and 2 have accuracy 0.53 and 0.9, respectively. That is,

$$f_1{}^*(x,1) = 0.7, \quad f_2{}^*(x,1) = 0.4$$

$$Acc_1(1) = 0.53, \quad Acc_2(1) = 0.9$$

Compute the corresponding BBAs of each classifier.

$$m_1(\Theta) = 1 - Acc_1(1) = 0.47, \quad m_2(\Theta) = 1 - Acc_2(1) = 0.1$$

$$m_1(1) = f_1{}^*(x,1) * Acc_1(1) = 0.371, \quad m_2(1) = f_2{}^*(x,1) * Acc_2(1) = 0.36$$

Then apply the fusion methods.

- Combining two classifiers by the weighted sum gives

$$w(1) = \frac{Acc_1(1) * f_1{}^*(x,1) + Acc_2(1) * f_2{}^*(x,1)}{Acc_1(1) + Acc_2(1)} \ = \ 0.5112 \ > \ 0.5.$$

Decision: Assign label 1 to the example.

- For DST-Fusion, compute K_{12} and combine evidence.

$$K_{12} = 1 - (m_1(1) * m_2(\bar{1}) + m_1(\bar{1}) * m_2(1)) = 0.7424$$

$$m(1) = \frac{m_1(1) * m_2(1) + m_1(1) * m_2(\Theta) + m_1(\Theta) * m_2(1)}{K_{12}} \ = \ 0.4578$$

$$m(\bar{1}) = \frac{m_1(\bar{1}) * m_2(\bar{1}) + m_1(\bar{1}) * m_2(\Theta) + m_1(\Theta) * m_2(\bar{1})}{K_{12}} = 0.4789$$

$$Bel(1) = m(1) < m(\bar{1}) = Bel(\bar{1})$$

Decision: Do not assign label 1 to the example.

In this example, DST-Fusion chooses to follow classifier 2 and not to label the example with class 1 because classifier 2 has very high accuracy. Quite the opposite, the weighted sum follows the prediction of classifier 1 although classifier 2 is much more accurate than classifier 1.

To conclude, even though the experiments show DST-Fusion and the weighted sum methods perform roughly the same on EUROVOC data, from the above numerical examples, we can see that the decisions made by DST-Fusion are fundamentally different from that by weighted sum. Weighted sum tends to base its decision on the confidence in the label of the classifiers, whereas the DST-Fusion bases its decision on the accuracy of classifiers. Following the Dempster-Shafer theory of evidence, the inference mechanism of DST-Fusion is analogous to human reasoning process. It seems more logical and sensible that we should believe in what the more accurate classifier says. How much confidence the classifier has in a label does not help if the classifier is not accurate. Therefore, with this analogy, we are in favor of DST-Fusion and believe it should provide more trusted decisions than the weighted sum.

CHAPTER 8

Partitioning a Feature Space

We have presented a strategy to handle classification problems of multi-label data in a high dimensional feature space. The very first step in exploiting the proposed algorithm is to partition the large feature set into a number of subsets. Despite that, the discussion on the feature set generation issue have been pushed aside, and in the experiments of earlier chapters we plainly randomly separated the feature set to generate feature subsets. In this last part of the dissertation we will look briefly into this pending issue with some experiments. Would the ensemble of subclassifiers yield accuracy at an acceptable level with simple random grouping of features? Or a sophisticated method to generate a superior partition of the feature space is needed?

Admittedly features are the key of the whole classification process. However, we will not go into the feature optimization topic here. Rather, it is assumed that features in the entire large feature set have already gone through feature selection procedure and we only need to divide features into groups from which subclassifiers can be induced. We consider comparing two different arrangements of features in feature subsets; one is the non-overlap where any two feature subsets are disjoint, and the other is the overlap where different feature subsets have some features in common so that they are not disjoint.

Concerning feature space partitioning, another question that has come up is whether the mutual relations among features that might be weakened or eliminated by the partitioning has any effect on the end result of the proposed fusion method. We will investigate this point with the help of the principal component analysis to identify the relationship between features.

8.1 Common Features in Feature Subsets

There is a number of studies related to the use of different feature subsets for each classifier. Since feature subsets will have only a partial view of each data point, diversity is very important in ensemble generation. Integrating classifications of base classifiers trained on too similar ensemble members will show no improvement over any of the constituent members due to redundant information. Ho (1998) has shown that simple random selection of feature subsets is an effective technique for generating feature subsets. Chen and Ho (2000) used decision forests for text categorization where each tree in the forest is constructed in a randomly chosen feature subspace and different feature dimensions may be selected at each split. Random subspace method is effective if the data set size is small relative to its dimensionality

Greene et al. (2004) discussed and evaluated a variety of ensemble generation strategies including random subspacing, random selection from the original feature space with replacement, and algorithms based on clustering. They demonstrated that diversity among ensemble members is necessary, but not sufficient to yield an improved solution without the selection of a suitable fusion scheme.

In our original idea of using subclassifiers combination approach to circumvent computational difficulty due to the high dimensional feature space is to simply subdivide features in the feature set into smaller groups. If all feature subsets are disjoint, the total number of features we need to process under this approach will not increase. However this may not be an ideal partitioning if it does not produce accurate final classification. It has been shown that using overlapping informative subsets of features may help to improve the performance. Random subspacing and random selection of features with replacement are examples of methods that output feature subsets having some features in common. The total number of features to process will increase in this case. Since it will require longer processing time, we want to know if having common features in subsets improves the accuracy of the classification system. The two cases that we compare are overlapping and non-overlapping feature subsets. For non-overlapping subsets, we randomly select the desired number of features from the feature set without replacement and place them in each subset. In this way the feature subsets are mutually disjoint. For overlapping subsets, we want to be able to control the number of common features in subsets in order to study and understand the effect of common features better. Therefore instead of using random selection with replacement, we add 25% more features into each of the non-overlapping feature subsets. The additional features in each subset are randomly selected from those features in other subsets. So the resulting feature subsets are not disjoint; there are some features that are in more than one feature subset.

We experiment with EUROVOC data using 5 feature subsets of 4 different sizes, 100, 200, 400, and 800 features for non-overlap case, and 125, 250, 500, and 1,000 features for overlap case. In Figure 8.1, the average precisions of DST-Fusion with overlapping and non-overlapping feature subsets are inferior to the one of NoFusion. As anticipated, between the two DST-Fusion results, we get slightly higher average precision from using overlapping feature subsets, but it comes at the price of longer CPU time. Still, both of them use much less time than NoFusion. From these results we may conclude that it is advantageous to use common features in DST-Fusion to improve classification accuracy and save a considerable amount of time on data processing.

8.2 Mutual Relations among Features

In any set of features, it is not unusual to find some of available features having mutual relationship, and it may be necessary that the features that come from multiple relations must be joined together in order to have enough power to distinguish classes. Since we generate feature subsets by randomly dividing the feature set, the mutual relations among features would be disturbed. We are interested in learning if there is any consequence of such division that may contribute to the degradation in the performance of DST-Fusion when compared to NoFusion where all features are together.

We study this issue on EUROVOC data with 1,000 and 2,000 features and RCV1-v2 data with 2,000 features. The first thing is to identify features that are in the same relationship. To accomplish this task, we consider two methods, cluster analysis and principal component analysis.

Cluster analysis seeks to identify homogeneous subgroups of cases in the data, i.e., features that are similar or share some common traits will be grouped together. There are several ensemble generation techniques that are clustering-based. However, the application of cluster analysis on our two data sets failed to generate feature clusters in the sizes appropriate for further fusion. For example, the 2,000 features of RCV1-v2 data were grouped by k-mean clustering into one large cluster of 1,632 features, and four small clusters of 217, 88, 39, and 24 features.

Principal component analysis (PCA) is a well known method for dimensionality reduction of the feature space (Johnson and Wichern 2002). By analyzing the covari-

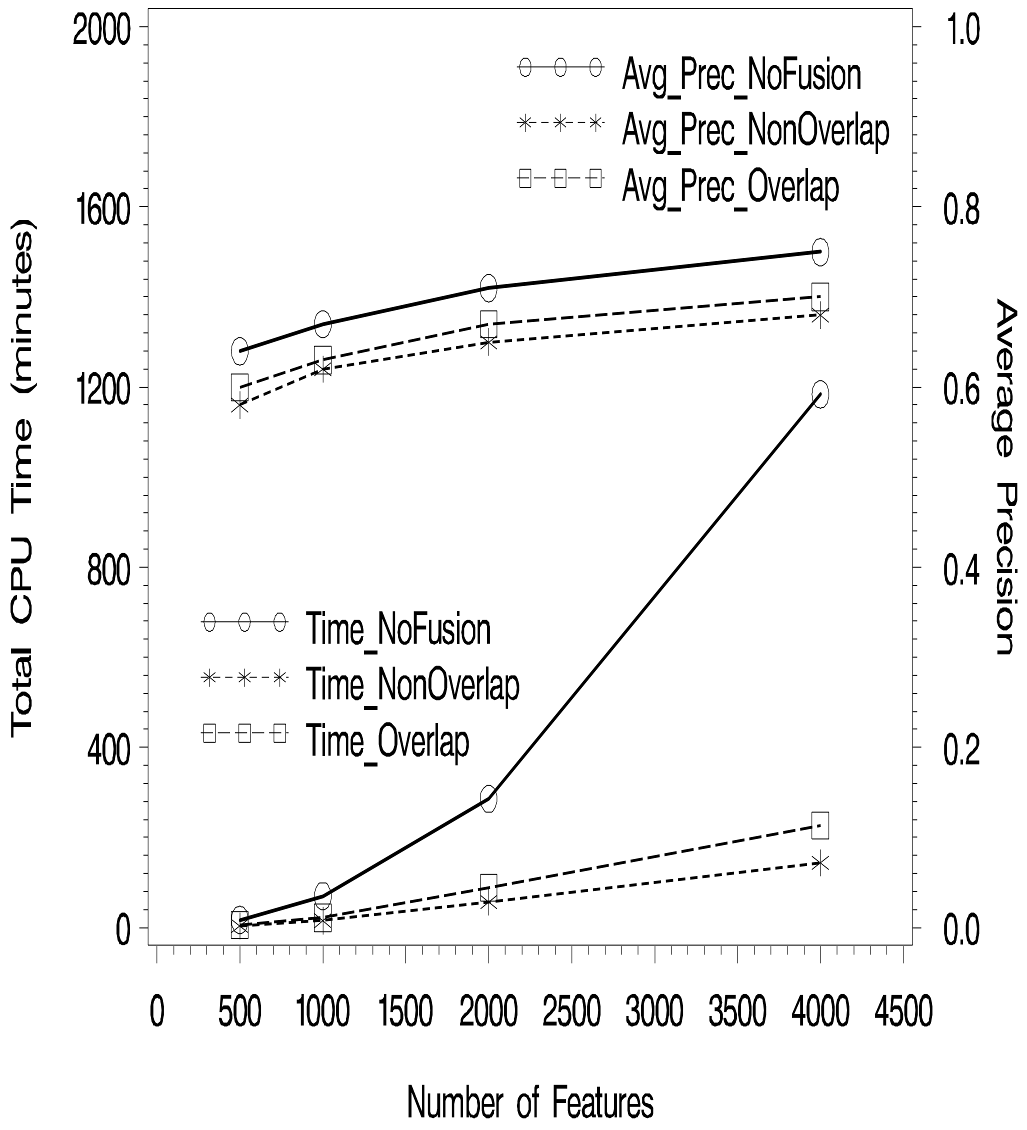


Figure 8.1: Total CPU time and average precision of two DST-Fusion cases, overlap and non-overlap, and NoFusion when varying number of features, EUROVOC data and 5-fold CV.

ance structure of data, a set of principal components are obtained. When the first few principal components corresponding to large eigenvalues of the covariance matrix can explain most of the total variation in the data, these few principal components can replace all of the original features in further analysis without much loss of information. However, our purpose of exploiting PCA here is different from what have been explained. We will make use of the last few principal components that correspond to very small eigenvalues to explore the linear dependence among features. This is an effective method to detect the presence of multicollinearity in regression analysis (Kutner et al. 2004).

For each of the data sets, we use the last five principal components to identify the features that have strong linear relationship (see details in Appendix E). Then we have 5 sets of features where features within each set are mutually dependent. For comparison purpose, extra features are randomly selected to be added in these sets so that each of them has the specified number of features. We apply DST-Fusion on these five sets of features. The number of features in each subset is 500, when the total number of features is 2,000, and 250 when the total number of features is 1,000. The number of boosting rounds is 10% of the number of features for EUROVOC data and 20% for RCV1-v2 data.

Presented in Tables 8.1 and 8.2 are the micro- and macro-average F_1 and the average precision. As before, NoFusion is apparently better than DST-Fusion on the average precision measure. There is no obvious difference between the two formations, random or PCA, of feature subsets, except the case of 2,000 features of EUROVOC data where PCA has lower micro-average F_1 than others.

These experiments tell us that random selection of features for feature subsets gives equally good performance as using PCA to form feature subsets. Random selection is preferred as it is easier. As a matter of fact, if a group of features have very

Table 8.1: Performance of NoFusion vs. DST-Fusion when features in feature subsets are (1) randomly selected and (2) from PCA, EUROVOC data. Bold items indicate significant difference from others at 0.05 level.

#Features	Method	Micro F	Macro F	Average Precision
1000	NoFusion	0.456 ± 0.020	0.372 ± 0.044	**0.669** ± 0.006
	Random	0.462 ± 0.020	0.294 ± 0.023	0.628 ± 0.007
	PCA	0.425 ± 0.026	0.282 ± 0.020	0.617 ± 0.013
2000	NoFusion	0.510 ± 0.015	0.447 ± 0.034	**0.710** ± 0.006
	Random	0.503 ± 0.012	0.369 ± 0.025	0.669 ± 0.009
	PCA	**0.469** ± 0.005	0.357 ± 0.029	0.648 ± 0.004

Table 8.2: Performance of NoFusion vs. DST-Fusion when features in feature subsets are (1) randomly selected and (2) from PCA, RCV1-v2 data. Bold items indicate the method is significantly better at 0.05 level.

Method	Micro F	Macro F	Average Precision
NoFusion	**0.675** ± 0.010	0.339 ± 0.010	**0.806** ± 0.005
Random	0.645 ± 0.007	0.330 ± 0.019	0.788 ± 0.004
PCA	0.644 ± 0.006	0.331 ± 0.008	0.787 ± 0.005

strong mutual relationship, it should not be necessary to use all features in that group together. Features not present in the feature subset will find their representatives in the subset. This is similar to what happens in the analysis of linear models with multicollinearity problem. Multicollinearity does not affect goodness of fit of a model. It merely makes the interpretation of the model harder. In other words, the prediction is still accurate.

Though the knowledge of mutual relations among features does not seem to help in the above experiments, we have tried another different way of utilizing the information about mutual relations. Theoretically, a group of independent classifiers improve upon the single best classifier when majority vote combination is used. However, Kuncheva et al. (2000) and Kuncheva et al. (2002) suggested independent classifiers may not be the best choice. They showed that dependent classifiers tend to offer a dramatic improvement over the individual accuracy, and in general, negative dependence is preferable. The negatively dependent classifiers will learn different aspects of training data, so that ensembles of classifiers can search in a wide solution space (Ryu and Sung-Bae 2002). They can intelligently complement each other, e.g., in the case when a classifier has difficulty categorizing an example, the other classifiers may be able to categorize it correctly. Quite the contrary, Shipp and Kuncheva (2002) indicated that dependent set of classifiers may be either better or worse, and although diversity is a desirable characteristic of feature subsets, diversity can be both beneficial or harmful. In any case, it is agreeable that classifier combination will result in improved classification performance if each separate feature subset contains some amount of useful and complementary information, and provided the combination scheme can exploit it.

To see if negatively correlated feature subsets will improve the performance of DST-Fusion, we conduct an experiment with RCV1-v2 data. We separate 2,000 fea-

Table 8.3: Performance of NoFusion vs. DST-Fusion when feature subsets are from (1) random selection, (2) PCA, and (3) negative correlation, RCV1-v2 data. Bold items indicate the method is significantly better at 0.05 level.

Method	Micro F	Macro F	Average Precision
NoFusion	**0.675** ± 0.010	0.339 ± 0.010	**0.806** ± 0.005
Random	0.645 ± 0.007	0.330 ± 0.019	0.788 ± 0.004
PCA	0.644 ± 0.006	0.331 ± 0.008	0.787 ± 0.005
NegCorr	0.636 ± 0.005	0.316 ± 0.014	0.779 ± 0.007

tures into 5 groups; two groups contain features that are positively correlated, another two groups also contain features positively correlated among themselves but have negative correlation with features in the first two groups, and the last group contains features that have very weak correlation with features in other groups. Table 8.3 shows the results of DST-Fusion from negative dependence feature subsets compared to previous results of random selection and PCA feature subsets as well as NoFusion. It is seen that DST-Fusion with random selection of feature subsets still compare favorably to other ensemble generation methods. The negative correlation does not do well perhaps because features in this data set do not exhibit very strong relationship. The largest pairwise correlation coefficient is found to be less than 0.15 in magnitude.

CHAPTER 9

Conclusion and Future Work

The dissertation has shown how text categorization, a major task in information retrieval, can benefit from machine learning techniques. The main point to remember is that text categorization differs from traditional classification problems in that each document can belong to more than one category at the same time—each example can have more than one class label. The dissertation therefore focused on the relatively new problem of induction from multi-label examples. The experience made by the many authors mentioned in the survey of literature is that this type of induction is computationally very costly, sometime prohibitively so. The research reported in this dissertation has resulted in a solution that significantly reduced these costs, paying for this considerable speedup by modestly impaired classification performance.

9.1 Summary and Contributions

A critical aspect of the domain that inspired this work is that the number of features describing each document is very high. This is serious in view of the fact that many existing algorithms need time that is exponential, or at least supralinear, in the number of features. The solution proposed in this dissertation reduces these computational costs by the use of a so-called *ensemble* approach. The idea is to induce several subclassifiers, each from a different subset of the features, and then combine ("fuse") the recommendation of these subclassifiers when applying them to the classification of testing examples.

Here are the major contributions of the work reported in this dissertation.

- The author designed and implemented a classifier induction system, DST-Fusion, for the induction of multi-label classifiers. The system proves particularly efficient when applied to large sets of data. Its essential principle is an ensemble of classifiers induced independently from feature subsets that are much smaller than the original feature set. This speeds up induction techniques whose computational complexity is superlinear in the number of features. The developed approach has two major aspects: induction of subclassifiers and fusion of subclassifiers. In the experiments reported here, the method made the baseline induction algorithm, *AdaBoost.MH* much faster, and the final predictive accuracy was not much worse than that observed in the case of learning from the complete set of features combined. The method is scalable in the sense that it does not collapse with the growing number of features.

- The author developed a method to combine the outputs of subclassifiers using a scheme based on the principles of the Dempster-Shafer theory. Unlike some other classifier combination methods, this scheme was shown to outperform any of the individual classifiers it combines.

- The author modified three traditional voting methods for combining single-label classifiers to make them handle multi-label domains. Extensive experiments have evaluated the differences in the decisions made by two classifier combination methods: the weighted sum and the Dempster-Shafer's evidence combination.

- The author compared the performance of four induction algorithms: *AdaBoost.MH, ADTree, C4.5*, and k-NN. From these, *AdaBoost.MH* was found to be much faster than the others. Its accuracy was acceptable, though not the best.

DST-Fusion clearly outperformed simple majority voting and weighted majority voting. Experimental results indicated that its performance was comparable also to that of the weighted sum rule, but in some cases seemed to give more sensible prediction.

Generally speaking, fusion of subclassifiers induced from feature subsets underperforms a "full-scale" classifier induced from all features combined. However, by accepting the small loss in accuracy, fusion has substantially reduced computational costs. Experiments indicate that the performance of the fusion method can be further improved by fine-tuning such as increasing the number of boosting rounds of the baseline induction algorithm, *AdaBoost.MH*. This, of course, increases the computation time, thus reducing the main advantage of fusion. Obviously, a major tradeoff is involved here.

Another advantage of the developed system is that each subclassifier is induced from a small feature subset, and therefore the memory requirements of each separate run are relatively small. Since inductions from different feature subsets can be handled independently of each other, the method can easily be parallelized. Thus further speeding up the induction process.

The author has also investigated the effect of feature subset selection on *DST-fusion*'s performance. A few different ways of generating these feature subsets were evaluated. It turned out that a simple random selection with some overlapping features worked adequately and was not worse than more complicated methods.

9.2 Future Research Direction

The research has indicated promising performance of the fusion approach in domains that call for induction of classifiers from very large sets of multi-label examples. The following natural extensions perhaps deserve further investigation.

- A basic component of the classifier induction system is the baseline induction algorithm. In this particular work, the *AdaBoost.MH* was chosen to this end. However, this may not necessarily be the best choice. More than likely, using a different algorithm for baseline induction might lead to higher performance. An interesting candidate for baseline induction algorithm is *SVM-Perf*. *SVM-Perf* is a support vector machine (SVM) that was improved by Joachims (2006) to reach the accuracy of the standard SVM, but it runs faster. SVM has the ability to generalize well in high dimensional feature space, and thus eliminates the need for feature selection. This makes it attractive for text categorization. In addition, in contrast to *AdaBoost.MH* where the number of boosting rounds has high impact on performance, SVM is less dependent on parameter setting. According to Joachims (2003), SVM promises a good prediction performance. Therefore it would be interesting to see how *DST-Fusion* will perform when *SVM-Perf* is used as the baseline induction algorithm.

- The weighted-sum rule turned out to have comparable performance to that of *DST-Fusion*, but sometimes each of them assigned very different labels to the same example. Comparing the two methods, *DST-Fusion* has slightly higher recall, and the weighted sum has slightly higher precision. It would be of interest to examine in detail the circumstances under which one method should be given preference over the other. Ideally, the two methods might be used to complement each other, and thus provide more reliable conclusion. For example, another level of fusion may be implemented to combine the results of both methods.

- In this work the question of the possible impact of feature selection methods was more or less left aside. Nevertheless, the potential of feature-selection methods to improve classification performance has already been recognized in general classification problems. However, a closely related issue to feature selection that should be also considered is how to generate the feature subsets. While traditional feature selection algorithms aim to find the best set of features relevant to the learning task and induction algorithm, the ensemble feature selection task also seeks to find a set of feature subsets that will promote diversity among the base classifiers. For example, if a feature-selection technique makes the induced subclassifiers complement each other in the spirit of the boosting algorithms, the accuracy of *DST-Fusion* is likely to improve. One possibility is to search for good feature subsets by clustering algorithm.

APPENDIX A

Validity of the Proposed BBA Calculation

Let Θ be a frame of discernment. Any basic belief assignment of a proposition $A \subseteq \Theta$, $m(A) \in [0, 1]$, must satisfy the following conditions:

$$m(\emptyset) = 0 \tag{A.1}$$

$$\sum_{A \subseteq \Theta} m(A) = 1 \tag{A.2}$$

We will show that the calculation of the BBAs by the procedure described in Section 5.3.1 produces BBAs that satisfy the above conditions:

Let us denote by $P(c)$ the prior probability that an example belongs to class c, and let $P(c|c')$ denote the conditional probability that the example is classified as belonging to class c when, in fact, the correct class label is c':

$$P(c|c') = P(\text{Classify an example as class } c | \text{True class of the example is } c')$$

For a single label c, we require that

$$P(\text{True class is } c) + P(\text{True class is not } c) = 1.$$

The first of these terms can be re-written as follows:

$$\begin{aligned}
P(\text{True class is } c) &= P(\text{True class is } c \text{ and classify as class } c) \\
&\quad + P(\text{True class is } c \text{ and classify as class not } c) \\
&= P(\text{Classify as class } c|\text{True class is } c)P(\text{True class is } c) \\
&\quad + P(\text{Classify as class not } c|\text{True class is } c)P(\text{True class is } c) \\
&= P(c|c)P(c) + P(\bar{c}|c)P(c)
\end{aligned}$$

Doing the same for the second term, we obtain the following relationship:

$$P(\text{True class is not } c) = P(c|\bar{c})P(\bar{c}) + P(\bar{c}|\bar{c})P(\bar{c})$$

Putting the two terms together then establishes the following equality:

$$P(c|c)P(c) + P(\bar{c}|c)P(c) + P(c|\bar{c})P(\bar{c}) + P(\bar{c}|\bar{c})P(\bar{c}) = 1.$$

From here, we conclude that

$$P(\bar{c}|c)P(c) + P(c|\bar{c})P(\bar{c}) \quad \leq \quad 1.$$

and, therefore,

$$m(\Theta) \quad = \quad P(c|\bar{c})P(\bar{c}) + P(\bar{cl}|c)P(c) \quad \leq \quad 1.$$

Finally, we obtain the following:

$$\begin{aligned}
m(\Theta) + m(c) + m(\bar{c}) \quad &= \quad [P(c|\bar{c})P(\bar{c}) + P(\bar{c}|c)P(c)] \\
&\quad + \; f^*(x,c) \times [1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)] \\
&\quad + \; [1 - f^*(x,c)] \times [1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)] \\
&= \quad [P(c|\bar{c})P(\bar{c}) + P(\bar{c}|c)P(c)] \\
&\quad + \; [1 - P(c|\bar{c})P(\bar{c}) - P(\bar{c}|c)P(c)] \\
&= \quad 1.
\end{aligned}$$

This completes the proof.

APPENDIX B

Performance of DST-Fusion vs. NoFusion

In the experiments of Section 6.2.1, we studied the effect of the number of features on the performance of DST-Fusion and NoFusion using the simplified EUROVOC database. We present in this appendix the detailed results. Four different sizes of feature set were used: 500, 1,000, 2,000, and 4,000 features. For NoFusion, we ran the AdaBoost.MH on each of the feature set. For DST-Fusion, in each case, the feature set was divided into five equally-sized subsets with 20% of the features overlapping with other subsets and then AdaBoost.MH was applied to feature subsets. AdaBoost.MH trained these data with the number of boosting rounds equal to 10% of the number of features.

Table B.1 shows five performance measurements that were estimated from 5-fold CV for NoFusion and DST-Fusion as well as each of the 5 individual subclassifier. No matter which feature size, DST-Fusion is not as good as NoFusion, but it has a higher performance than individual subclassifiers.

Table B.1: Multi-label performance measurements for DST-Fusion of 5 subclassifiers and NoFusion compared to individual subclassifiers. (Estimated from 5-fold CV. Bold items indicate that NoFusion is significantly better at 0.05 level.)

# Features	Average Precision	Coverage	Hamming Loss	One Error	Ranking Loss
500					
SET1	0.56 ± .008	10.20 ± .064	0.11 ± .001	0.41 ± .016	0.16 ± .003
SET2	0.57 ± .009	10.14 ± .103	0.11 ± .002	0.40 ± .019	0.15 ± .004
SET3	0.57 ± .008	10.13 ± .075	0.11 ± .001	0.40 ± .016	0.15 ± .003
SET4	0.56 ± .010	10.19 ± .097	0.11 ± .001	0.43 ± .017	0.16 ± .004
SET5	0.56 ± .008	10.28 ± .118	0.11 ± .001	0.42 ± .014	0.16 ± .004
DST	0.60 ± .009	9.59 ± .071	0.11 ± .001	0.35 ± .021	0.14 ± .004
NoFusion	**0.64** ± .007	**8.77** ± .094	**0.10** ± .002	0.32 ± .018	**0.12** ± .002
1000					
SET1	0.58 ± .007	9.86 ± .101	0.11 ± .001	0.39 ± .016	0.15 ± .003
SET2	0.59 ± .007	9.72 ± .085	0.11 ± .002	0.37 ± .015	0.14 ± .003
SET3	0.57 ± .005	9.76 ± .076	0.11 ± .001	0.41 ± .016	0.15 ± .002
SET4	0.59 ± .006	9.50 ± .038	0.11 ± .001	0.39 ± .006	0.14 ± .003
SET5	0.59 ± .008	9.78 ± .092	0.11 ± .002	0.37 ± .012	0.14 ± .003
DST	0.63 ± .007	9.05 ± .045	0.11 ± .001	0.32 ± .019	0.12 ± .002
NoFusion	**0.67** ± .006	**8.15** ± .077	**0.10** ± .001	0.29 ± .011	**0.11** ± .002
2000					
SET1	0.62 ± .008	9.12 ± .084	0.11 ± .002	0.35 ± .015	0.13 ± .003
SET2	0.62 ± .005	9.04 ± .036	0.10 ± .001	0.36 ± .004	0.13 ± .002
SET3	0.62 ± .005	9.00 ± .055	0.10 ± .001	0.35 ± .014	0.13 ± .002
SET4	0.61 ± .010	9.21 ± .100	0.11 ± .002	0.35 ± .022	0.13 ± .003
SET5	0.62 ± .009	9.13 ± .093	0.10 ± .002	0.35 ± .021	0.13 ± .004
DST	0.67 ± .009	8.35 ± .071	0.10 ± .002	0.27 ± .016	0.11 ± .003
NoFusion	**0.71** ± .006	**7.49** ± .042	**0.09** ± .002	0.24 ± .015	**0.09** ± .002
4000					
SET1	0.65 ± .007	8.62 ± .075	0.10 ± .001	0.31 ± .017	0.12 ± .002
SET2	0.64 ± .008	8.76 ± .077	0.10 ± .002	0.33 ± .024	0.12 ± .002
SET3	0.65 ± .008	8.50 ± .056	0.10 ± .001	0.31 ± .016	0.12 ± .002
SET4	0.65 ± .006	8.55 ± .051	0.10 ± .001	0.31 ± .012	0.12 ± .003
SET5	0.64 ± .008	8.67 ± .051	0.10 ± .001	0.32 ± .018	0.12 ± .003
DST	0.70 ± .008	7.72 ± .031	0.10 ± .002	0.23 ± .017	0.10 ± .003
NoFusion	**0.75** ± .008	**6.89** ± .041	**0.08** ± .002	0.20 ± .016	**0.08** ± .002

APPENDIX C

Number of Boosting Rounds in Experiments

The number of boosting rounds for AdaBoost.MH used in most of our experiments with EUROVOC data was 10% of the number of features. As AdaBoost.MH selects one feature at each round, the 10% of the number of features would allow both NoFusion and DST-Fusion to utilize the information from approximately the same number of features out of the feature pool that is available to them when categorizing a document. This gives an advantage to NoFusion to select good features for weak hypotheses from a larger set of features, while DST-Fusion is restricted to selecting features from smaller feature subsets. Consequently, NoFusion has higher accuracy than DST-Fusion and inevitably longer run time. What would be interesting is if DST-Fusion uses the same number of boosting rounds as NoFusion and surely DST-Fusion will not be in a favorable position considering the CPU time, is DST-Fusion able to compete with NoFusion on accuracy?

To understand this point better, we do more experimentation on the simplified EUROVOC database. For NoFusion, we use all 1,000 features, and for DST-Fusion, we divide feature set into 5 subsets of equal size in 2 ways, non-overlapping subsets each having 200 features and 20% overlapping subsets each having 250 features. The number of boosting rounds is always set to be the same for both approaches, and we

Table C.1: Performance of NoFusion and DST-Fusion of 5 subclassifiers from overlapping and non-overlapping feature subsets. Both methods use the same number of boosting rounds.

#Boosting Rounds	Method	Micro F	Macro F	Average Precision	CPU Time (Minutes)
10	NoFusion	.369 ± .010	.173 ± .008	.580 ± .009	8.1
	DST_NonOvl	.426 ± .008	.228 ± .012	.588 ± .010	8.0
	DST_Ovl	.426 ± .006	.224 ± .009	.587 ± .008	10.0
50	NoFusion	.425 ± .028	.282 ± .024	.644 ± .007	35.7
	DST_NonOvl	.482 ± .004	.349 ± .022	.646 ± .007	35.2
	DST_Ovl	.486 ± .006	.378 ± .034	.651 ± .006	43.7
100	NoFusion	.456 ± .020	.372 ± .044	.669 ± .006	71.6
	DST_NonOvl	.510 ± .010	.416 ± .044	.665 ± .007	69.0
	DST_Ovl	.509 ± .007	.414 ± .031	.669 ± .006	86.5
250	NoFusion	.494 ± .006	.442 ± .028	.701 ± .004	174.7
	DST_NonOvl	.546 ± .011	.488 ± .034	.694 ± .004	170.6
	DST_Ovl	.550 ± .010	.489 ± .028	.698 ± .005	211.8

vary this number as 10, 50, 100, and 250. Table C.1 compares NoFusion to DST-Fusion for overlap and non-overlap cases. Learning from overlapping feature subsets usually produces somewhat better classifiers than learning from non-overlapping subsets. DST-Fusion is superior to NoFusion on micro- and macro-average F_1 On average precision measure, DST-Fusion also tends to be better. As far as the CPU time is concerned, DST-Fusion with non-overlapping feature subsets uses a little less total CPU time than NoFusion, and DST-Fusion with overlapping subsets uses more time than NoFusion due to more features in total need to be processed.

APPENDIX D

Profile of RCV1-v2 Data

RCV1-v2 is a text categorization test collection converted from Reuters Corpus Volume 1 (RCV1) after removing various errors by Lewis et al. (2004). It consists of 804,414 newswires stories based on data released by Reuters, Ltd. Each news story (document) has been categorized according to its content and identified by a unique document ID. A total of 47,236 features are extracted from the documents and indexed.

Five training sets and five test sets from RCV1-v2 collection [1] are selected for use in our experiments. There are 103 topic categories, 101 with one or more positive training examples on training sets. Figure D.1 shows the distributions of topic categories and Figure D.2 shows the multi-label character of documents in each of the data sets. Obviously, all data sets possess almost identical multi-label characteristics.

[1] http://mlkd.csd.auth.gr/multilabel.html#Datasets

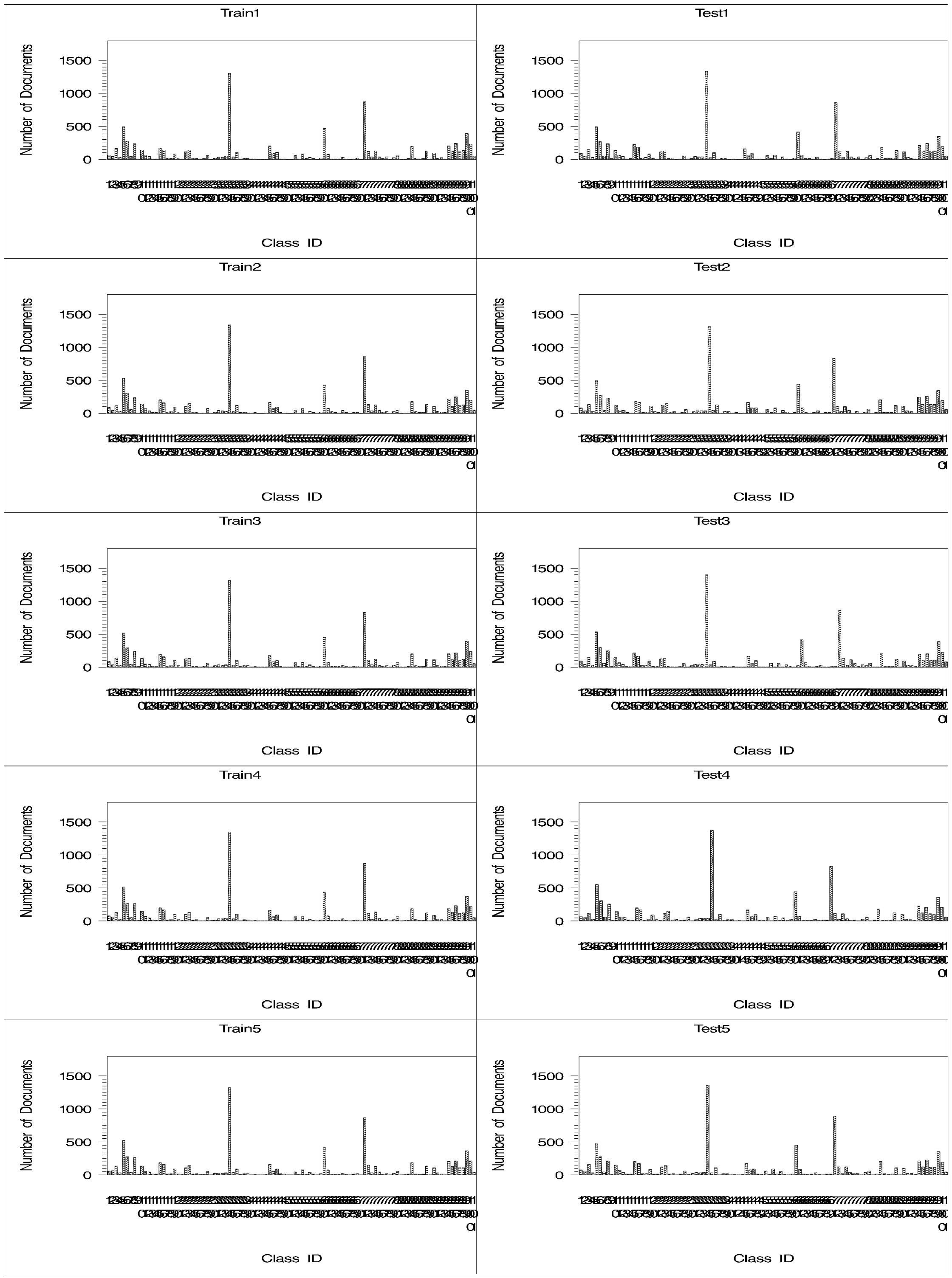


Figure D.1: Class distributions of documents in each of five RCV1-v2 experimental training data sets and five test data sets, non-disjoint classes. 3,000 documents in each set.

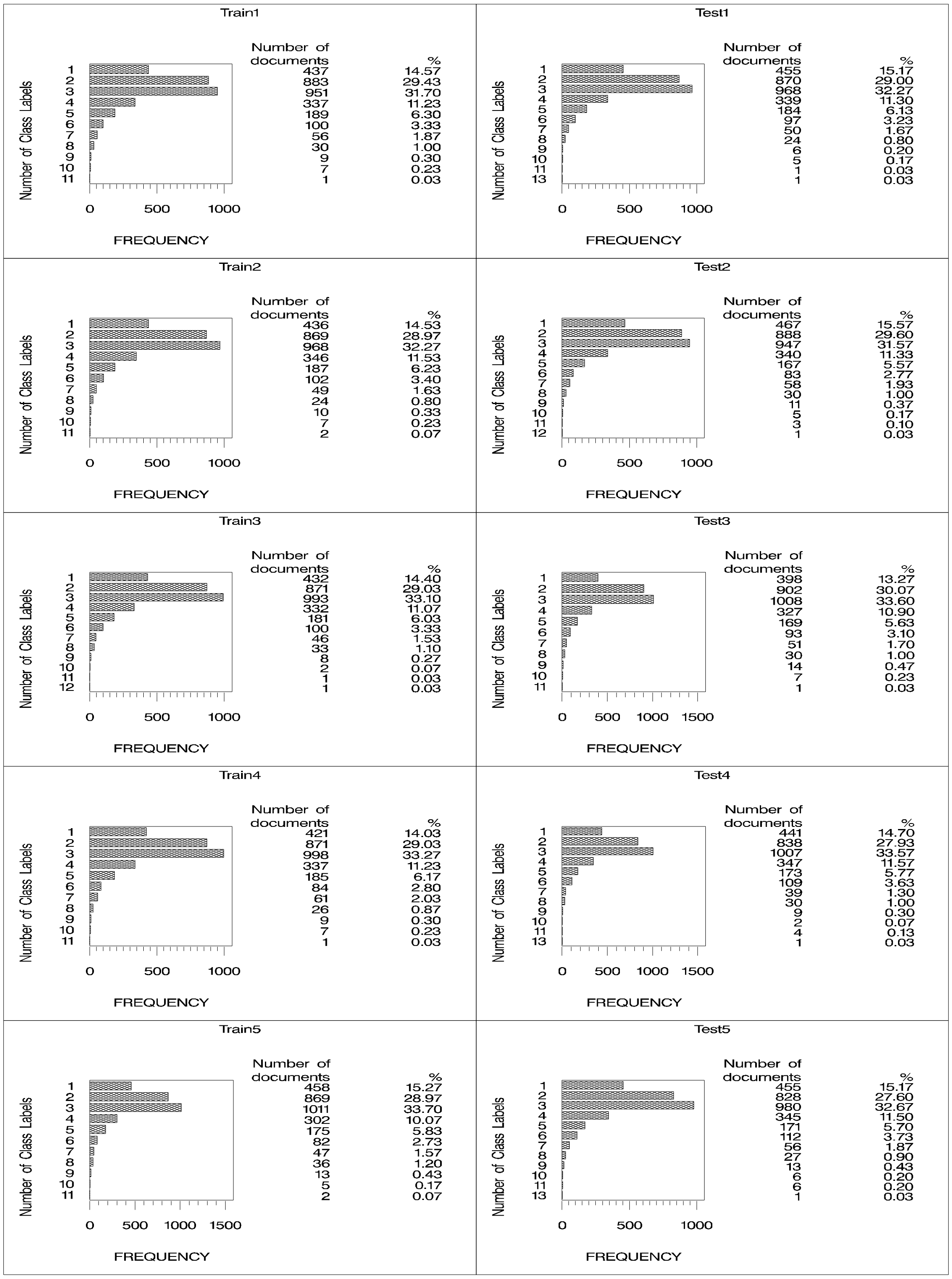


Figure D.2: Number of documents in each of five RCV1-v2 experimental training data sets and five test data sets having different numbers of class labels.

APPENDIX E

Linear Relationship Between Features

The principal component analysis (PCA) is concerned with explaining the variance-covariance structure of a set of variables (or features) through a few new variables which are linear combinations of the original variables. By creating new data of this smaller set of new variables, the dimensionality of original data is reduced (Johnson and Wichern 2002). Dimensionality reduction is usually the main purpose for exploiting PCA. Another less common use of PCA is in identifying the source of multicollinearity in data.

Principal components are particular linear combinations of the p random variables x_1, x_2, …, x_p, with three important properties: (1) the principal components are uncorrelated, (2) the first principal component has the highest variance, the second principal component has the second highest variance, and so on, and (3) the total variation in all the principal components combined equal to the total variation in the original variables x_1, x_2, …, x_p. The new variables with such properties can be easily obtained from the eigenanalysis of the covariance matrix or the correlation matrix of x_1, x_2, …, x_p.

Let the original data $\mathbf{X}$ be an $n \times p$ data matrix of n observations on each of the p variables x_1, x_2, …, x_p and let $\mathbf{S}$ be a $p \times p$ sample covariance matrix of x_1, x_2,

…, x$_p$. If (λ_1, $\mathbf{e}_1$), (λ_2, $\mathbf{e}_2$), …, (λ_p, $\mathbf{e}_p$) are the p eigenvalue-eigenvector pairs of the matrix $\mathbf{S}$, then the i-th principal component is

$$u_i \;=\; \mathbf{e}_i^{'}(\mathbf{x}-\bar{\mathbf{x}}) \;=\; e_{i1}(x_1-\bar{x}_1)+e_{i2}(x_2-\bar{x}_2)+\ldots+e_{ip}(x_p-\bar{x}_p), \quad i=1,2,\ldots,p$$

where

$\lambda_1 \;\geq\; \lambda_2 \;\geq \ldots \geq\; \lambda_p \geq 0,$

$\mathbf{e}_i^{'} \;=\; (e_{i1}, e_{i2}, \ldots, e_{ip})$ is the i-th eigenvector,

$\mathbf{x}^{'} \;=\; (x_1, x_2, \ldots, x_p)$ is any observation vector of the p variables, x$_1$, x$_2$, …, x$_p$,

and

$\bar{\mathbf{x}}^{'} \;=\; (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_p)$ is the sample mean vector of the p variables, x$_1$, x$_2$, …, x$_p$.

The i-th principal component has sample variance λ_i and the sample covariance of any pair of principal components is 0. In addition, if s$_{ii}$ is the sample covariance of the variable x$_i$, then the total sample variance in all variables x$_1$, x$_2$, …, x$_p$ is

$$\sum_{i=1}^{p} s_{ii} \;=\; \lambda_1+\lambda_2+\ldots+\lambda_p$$

which is the total sample variance in all the principal components. This means that instead of working with the original variables x$_1$, x$_2$, …, x$_p$, we can work with the principal components and get the same result. There is no loss of information since all of the variation in the original data is accounted for by the principal components.

PCA can be carried out on the $p \times p$ sample correlation matrix $\mathbf{R}$ of the variables x$_1$, x$_2$, …, x$_p$ in the same fashion as with the covariance matrix. If (λ_1, $\mathbf{e}_1$), (λ_2, $\mathbf{e}_2$), …, (λ_p, $\mathbf{e}_p$) are the p eigenvalue-eigenvector pairs of the matrix $\mathbf{R}$, then the i-th principal component takes the form

$$u_i \;=\; \mathbf{e}_i^{'}\mathbf{z} \;=\; e_{i1}z_1+e_{i2}z_2+\ldots+e_{ip}z_p, \quad i=1,2,\ldots,p \tag{E.1}$$

where

$\mathbf{z}' = (z_1, z_2, \ldots, z_p)$ is the vector of standardized observations defined as

$$z_k = \frac{(x_k - \bar{x}_k)}{\sqrt{s_{kk}}}, \quad k = 1, 2, \ldots, p$$

The principal components from the sample correlation matrix still have the same properties as before. That is, the i-th principal component has sample variance λ_i, the sample covariance of any pair of principal components is 0, and the total sample variance in all the principal components is

$$\lambda_1 + \lambda_2 + \ldots + \lambda_p = p$$

which is the total sample variance in all standardized variables z_1, z_2, ..., z_p.

The last few principal components represent linear functions of the original variables with minimal variance. If the variances are close to zero, it means these components vary very little around their mean values which is also zero. For example, the p-th principal component has variance $\lambda_p \approx 0$, we can approximate this last principal component by its mean, 0. That is, from Equation (E.1) we have

$$u_p = e_{p1}z_1 + e_{p2}z_2 + \ldots + e_{pp}z_p \approx 0, \tag{E.2}$$

or

$$e_{p1}\frac{(x_1 - \bar{x}_1)}{\sqrt{s_{11}}} + e_{p2}\frac{(x_2 - \bar{x}_2)}{\sqrt{s_{22}}} + \ldots + e_{pp}\frac{(x_p - \bar{x}_p)}{\sqrt{s_{pp}}} \approx 0. \tag{E.3}$$

This gives the approximate linear relationship among variables. We can simplify the relationship by discarding any variable i associated with small eigenvector coefficient e_{pi}. More linear relationships can also be found from other last few principal components in the same way.

Bibliography

Aas, K. and Eikvil, L. 1999. Text categorisation: A survey. Technical report, Norwegian Computing Center, June 1999.

Al-Ani, A. and Deriche, M. 2002. A new technique for combining multiple classifiers using the Dempster-Shafer theory of evidence. *Jour. of Artificial Intelligence Research 17*, 333–361.

Altinçay, H. 2005. A Dempster-Shafer theoretic framework for boosting based ensemble design. *Pattern Analysis & Applications 8,* 3 (Dec.), 287–302.

Aphinyanaphongs, Y., Tsamardinos, I., Statnikov, A., Hardin, D., and Aliferis, C. F. 2005. Text categorization models for high quality article retrieval in internal medicine. *Journal of the American Medical Informatics Association 12,* 2, 207–216.

Bahler, D. and Navarro, L. 2000. Methods for combining heterogeneous sets of classifiers. In *Proc. Natl. Conf. on Artificial Intelligence (AAAI) Workshop on New Research Problems for Machine Learning.*

Bell, D. A., Guan, J., and Bi, Y. 2005. On combining classifier mass functions for text categorization. *IEEE Trans. on Knowledge and Data Engineering 17,* 10 (Oct.), 1307–1319.

Bennett, P. N., Dumais, S. T., and Horvitz, E. 2005. The combination of text classifiers using reliability indicators. *Information Retrieval 8,* 1, 67–100.

Bi, Y., Bell, D. A., Wang, H., Guo, G., and Greer, K. 2004. Combining multiple classifiers using Dempster's rule of combination for text categorization. In *Proc. Int'l Conf. Modeling Decisions for Artificial Intelligence.* 127–138.

Bi, Y., McClean, S., and Anderson, T. 2005. Improving classification decisions by multiple knowledge. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence (ICTAI'05).* 340–347.

Blake, C. and Pratt, W. 2001. Better rules, few features: A semantic approach to selecting features from text. In *ICDM*. 59–66.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. 2004. Learning multi-label scene classification. *Pattern Recognition 37,* 9, 1757–1771.

Breiman, L. 2001. Random forests. *Machine Learning 45*, 5–32.

Cai, L. and Hofmann, T. 2003. Text categorization by boosting automatically extracted concepts. In *SIGIR*. 182–189.

Calvo, R. A., Lee, J.-M., and Li, X. 2004. Managing content with automatic document classification. *Journal of Digital Information 5*.

Challa, S. and Koks, D. 2004. Bayesian and dempstershafer fusion. *29*, 145–176.

Chen, H. and Ho, T. K. 2000. Evaluation of decision forests on text categorization. In *Proceedings of the 7th SPIE Conference on Document Recognition and Retrieval*, D. P. Lopresti and J. Zhou, Eds. SPIE - The International Society for Optical Engineering, San Jose, US, 191–199.

Chen, J., Zhou, X., and Wu, Z. 2004. A multi-label Chinese text categorization system based on boosting algorithm. In *Proc. IEEE Int'l Conf. on Computer and Information Technology (CIT'04)*. 1153–1158.

Clare, A. and King, R. D. 2001. Knowledge discovery in multi-label phenotype data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'01)*. Freiburg, Germany.

De Comité, F., Gilleron, R., and Tommasi, M. 2001. Learning multi-label alternating decision trees and applications. In *Proc. Conf. en Apprentissage Automatique (CAP'01)*. 195–210.

De Comité, F., Gilleron, R., and Tommasi, M. 2003. Learning multi-label alternating decision trees from texts and data. In *Proc. Int'l Conf. on Machine Learning and Data Mining (MLDM'03)*. 35–49.

Diao, L., Hu, K., Lu, Y., and Shi, C. 2002. Boosting simple decision trees with Bayesian learning for text categorization. In *Proc. World Congress on Intelligent Control and Automation*. Shanghai, P.R.China, 321–325.

Diplaris, S., Tsoumakas, G., Mitkas, P. A., and Vlahavas, I. P. 2005. Protein classification with multiple algorithms. In *Panhellenic Conference on Informatics*. 448–456.

Drucker, H. and Cortes, C. 1996. Boosting decision trees. In *Advances in Neural Information processing Systems 8*. 479–485.

Esuli, A., Fagni, T., and Sebastiani, F. 2006. Mp-boost: A multiple-pivot boosting algorithm and its application to text categorization. In *Proceedings of the 13th International Symposium on String Processing and Information Retrieval (SPIRE'06)*. Glasgow, UK.

Eyheramendy, S., Lewis, D. D., and Madigan, D. 2003. On the naive Bayes model for text categorization. In *Proc. Int'l Workshop on Artificial Intelligence and Statistics*.

Feldman, R. and Sanger, J. 2007. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, New York, NY, 410.

Fern, X. Z. and Brodley, C. E. 2003. Boosting lazy decision trees. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*. Washington, DC, USA, 178–185.

Forman, G. 2007. Feature selection for text classification. In *Computational Methods of Feature Selection*. CRC Press/Taylor and Francis Group.

Freund, Y. and Mason, L. 1999. The alternating decision tree learning algorithm,. In *Proc. Int'l Conf. on Machine Learning (ICML'99)*. Morgan Kaufmann, San Francisco, CA, 124–133.

Freund, Y. and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *Proc. Int'l Conf. on Machine Learning (ICML'96)*. 148–156.

Freund, Y. and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Jour. of Computer and System Sciences 55,* 1 (Aug.), 119–139.

Friedman, N., Geiger, D., and Goldszmidt, M. 1997. Bayesian network classifiers. *Machine Learning 29,* 2-3, 131–163.

Fürnkranz, J. 2002. Hyperlink ensembles: A case study in hypertext classification. *Information Fusion 3,* 4 (Dec.), 299–312.

Gao, S., Wu, W., Lee, C.-H., and Chua, T.-S. 2004. A MFoM learning approach to robust multiclass multi-label text categorization. In *Proc. Int'l Conf. on Machine Learning (ICML'04)*. 329–336.

Godbole, S. and Sarawagi, S. 2004. Discriminative methods for multi-labeled classification. In *Proc. Pacific-Asia Conference (PAKDD'04)*. 22–30.

Greene, D., Tsymbal, A., Bolshakova, N., and Cunningham, P. 2004. Ensemble clustering in medical diagnostics. In *CBMS '04: Proceedings of the 17th IEEE Symposium on Computer-Based Medical Systems (CBMS'04)*. IEEE Computer Society, Washington, DC, USA.

Grove, A. J. and Schuurmans, D. 1998. Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*. 692–699.

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

Han, E.-H., Karypis, G., and Kumar, V. 2001. Text categorization using weight adjusted k-nearest neighbor classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 53–65.

Ho, T. K. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20,* 8, 832–844.

Holmes, G., Pfahringer, B., Kirkby, R., Frank, E., and Hall, M. 2002. Multiclass alternating decision trees. In *Proc. European Conf. on Machine Learning (ECML'02)*. Springer Verlag.

Hsu, J. C. 1996. *Multiple Comparisons: Theory and Methods*. Chapman and Hall/CRC.

Ittner, D. J., Lewis, D. D., and Ahn, D. D. 1995. Text categorization of low quality images. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*. Las Vegas, US, 301–315.

Jain, G., Ginwala, A., and Aslandogan, Y. A. 2004. An approach to text classification using dimensionality reduction and combination of classifiers. In *IRI*. 564–569.

Joachims, T. 1998. Text categorization with support vector machines: learning with many relevant features. In *Proc. European Conf. on Machine Learning (ECML'98)*, C. Nédellec and C. Rouveirol, Eds. Number 1398. Springer Verlag, Heidelberg, DE, Chemnitz, DE, 137–142.

Joachims, T. 2003. *Learning to Classify Text Using Support Vector Machines.* Kluwer Academic Publishers, Norwell, MA.

Joachims, T. 2006. Training linear svms in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM.

Johnson, M. and Cipolla, R. 2005. Improved image annotation and labelling through multi-label boosting. In *Proc. the 2005 British Machine Vision Conference (BMVC '05)*.

Johnson, R. A. and Wichern, D. W. 2002. *Applied Multivariate Statistical Analysis*, 5 ed. Prentice Hall.

Kang, F., Jin, R., and Sukthankar, R. 2006. Correlated label propagation with application to multi-label learning. In *CVPR (2)*. 1719–1726.

Koller, D. and Sahami, M. 1996. Toward optimal feature selection. In *ICML*. 284–292.

Kubat, M. and Cooperson, M., J. 2001. A reduction technique for nearest-neighbor classification: Small groups of examples. *Intelligent Data Analysis 5*, 463–476.

Kuncheva, L., Skurichina, M., and Duin, R. 2002. An experimental study on diversity for bagging and boosting with linear classifiers. *Information Fusion 3,* 4, 245–258.

Kuncheva, L. I., Whitaker, C., Shipp, C., and Duin, R. 2000. Is independence good for combining classifiers? *Proc. 15th International Conference on Pattern Recognition (ICPR'00) 2*, 168–171.

Kutner, M. H., Nachtsheim, C. J., and Neter, J. 2004. *Applied Linear Regression Models*, 4 ed. McGraw Hill/Irwin.

Kwok, J. T. 1998. Automated text categorization using support vector machine. In *Proc. Int'l Conf. on Neural Information Processing (ICONIP'98)*. Kitakyushu, JP, 347–351.

Lafferty, J. and Wasserman, L. 2006. Commentary - challenges in statistical machine learning. *Statistica Sinica 16*, 307–322.

Lam, L. and Suen, C. Y. 1997. Application of majority voting to pattern recognition: An analysis of the behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics 27,* 5, 553–568.

Langley, P., Iba, W., and Thompson, K. 1992. An analysis of Bayesian classifiers. In *Proc. of the 10th Natl. Conf. on Artificial Intelligence.* AAAI Press and M.I.T. Press, 223–228.

Larkey, L. S. and Croft, W. B. 1996. Combining classifiers in text categorization. In *Proc. ACM SIGIR Conf. Research and Development in Information Retrieval*, H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, Eds. ACM Press, New York, US, Zürich, CH, 289–297.

LAUSER, B. AND HOTHO, A. 2003. Automatic multi-label subject indexing in a multilingual environment. In *ECDL*. 140–151.

LEWIS, D. D. 1992a. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of SIGIR-92, 15th ACM International Conference on Research and Development in Information Retrieval.* 37–50.

LEWIS, D. D. 1992b. Feature Selection and Feature Extraction for Text Categorization. In *Proceedings of Speech and Natural Language Workshop*. Morgan Kaufmann, San Mateo, California, 212–217.

LEWIS, D. D., YANG, Y., ROSE, T. G., AND LI, F. 2004. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research 5*, 361–397.

LI, B., LU, Q., AND YU, S. 2004. An adaptive k-nearest neighbor text categorization strategy. *ACM Trans. on Asian Language Information Processing (TALIP) 3*, 215–226.

LI, B., YU, S., AND LU, Q. 2003. An improved k-nearest neighbor algorithm for text categorization. In *Proceedings of the 20th International Conference on Computer Processing of Oriental Languages.*

LI, T. AND OGIHARA, M. 2003. Detecting emotion in music. In *Proc. of the International Symposium on Music Information Retrieval.* 239–240.

LIN, X., YACOUB, S. M., BURNS, J., AND SIMSKE, S. J. 2003. Performance analysis of pattern classifier combination by plurality voting. *Pattern Recognition Letters 24,* 12, 1959–1969.

LIU, H. AND MOTODA, H. 2002. On issues of instance selection. *Journal of Data Mining and Knowledge Discovery 6*, 115–130.

LU, X., WANG, Y., AND JAIN, A. K. 2003. Combining classifiers for face recognition. In *IEEE International Conference on Multimedia & Expo (ICME'03).* Vol. III. 13–16.

MCCALLUM, A. AND NIGAM, K. 1998. A comparison of event models for naive Bayes text classification. In *Proc. Workshop on Learning for Text Categorization (AAAI'98).*

MCCALLUM, A. K. 1999. Multi-label text classification with a mixture model trained by EM. In *Proc. of AAAI'99 Workshop on Text Learning.*

MICHALSKI, R. S., BRATKO, I., AND KUBAT, M. 1998. *Machine Learning and Data Mining: Methods and Applications.* John Wiley.

Nigam, K., McCallum, A. K., Thrun, S., and Mitchell, T. M. 2000. Text classification from labeled and unlabeled documents using EM. *Machine Learning 39,* 2/3, 103–134.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, CA.

Quinlan, J. R. 1996a. Bagging, boosting, and c4.5. In *Proc. of the 13th Nat'l. Conf. on Artificial Intelligence.* AAAI Press and the MIT Press, 725–730.

Quinlan, J. R. 1996b. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research 4*, 77–90.

Quinlan, J. R., Compton, P. J., Horn, K. A., and Lazarus, L. 1986. Inductive knowledge acquisition: a case study. In *Proceedings of the Second Australian Conference on applications of expert systems.* 183–204.

Rahman, A. F. R., Alam, H., and Fairhurst, M. C. 2002. Multiple classifier combination for character recognition: Revisiting the majority voting system and its variations. *Lecture Notes in Computer Science 2423*, 167 ff.

Ramasubramanian and Paliwal. 2000. Fast nearest-neighbor search algorithms based on approximation-elimination search. *PATREC: Pattern Recognition, Pergamon Press 33.*

Roli, F. and Giacinto, G. 2002. Design of multiple classifier systems.

Rose, T., Stevenson, M., and Whitehead, M. 2002. The reuters corpus volume 1 from yesterdays news to tomorrows language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation.* 827–832.

Ruiz, M. E. and Srinivasan, P. 1998. Automatic text categorization using neural networks. In *Advances in Classification Research: Proc. Classification Research Workshop (ASIS SIG/CR).* Vol. 8. Medford, New Jersey, 59–72.

Ruta, D. and Gabrys, B. 2000. An overview of classifier fusion methods. *Computing and Information Systems 7,* 1 (February), 1–10.

Ryu, J. and Sung-Bae, C. 2002. Gene expression classification using optimal feature/classifier ensemble with negative correlation. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN02).* Honolulu, Hawaii, 198–203.

Sarinnapakorn, K. and Kubat, M. 2007a. Combining subclassifiers in text classification: A dst-based solution and a case study. *IEEE Transactions on Knowledge and Data Engineering 19,* 12 (December), 1638–1651.

SARINNAPAKORN, K. AND KUBAT, M. 2007b. Induction from multi-label training examples in text categorization: Combining subclassifiers (a case study). In *Proc. of the 2007 International Conference on Artificial Intelligence (ICAI'07)*. CSREA Press, 351–357.

SARINNAPAKORN, K. AND KUBAT, M. 2008. Induction from multilabel examples in information retrieval systems. *Applied Artificial Intelligence 22,* 1 (January).

SCHAPIRE, R. E. 1990. The strength of weak learnability. *Machine Learning 5,* 2, 197–227.

SCHAPIRE, R. E. 1999. A brief introduction to boosting. In *IJCAI*. 1401–1406.

SCHAPIRE, R. E. AND SINGER, Y. 1999. Improved boosting using confidence-rated predictions. *Machine Learning 37,* 3, 297–336.

SCHAPIRE, R. E. AND SINGER, Y. 2000. BoosTexter: A boosting-based system for text categorization. *Machine Learning 39,* 2/3, 135–168.

SEBASTIANI, F. 2002. Machine learning in automated text categorization. *ACM Comput. Surv. 34,* 1, 1–47.

SEBASTIANI, F. 2006. Classification of text, automatic. In *The Encyclopedia of Language and Linguistics*, Second ed., K. Brown, Ed. Vol. 2. Elsevier Science Publishers, Amsterdam, NL, 457–463.

SHAFER, G. 1976. *A mathematical theory of evidence*. Princeton University Press.

SHEN, X., BOUTELL, M., LUO, J., AND BROWN, C. 2004. Multi-label machine learning and its application to semantic scene classification. In *Proc. Int'l Symposium on Electronic Imaging*. San Jose, CA.

SHIPP, C. A. AND KUNCHEVA, L. I. 2002. Relationships between combination methods and measures of diversity in combining classifiers. *Information Fusion 3,* 2, 135–148.

SU, C.-Y., LO, A., LIN, C.-C., CHANG, F., AND HSU, W.-L. 2005. A novel approach for prediction of multi-labeled protein subcellular localization for prokaryotic bacteria. In *2005 IEEE Computational Systems Bioinformatics Conference - Workshops (CSBW'05)*. IEEE Computer Society, Los Alamitos, CA, USA, 79–82.

SUN, A. AND LIM, E.-P. 2001. Hierarchical text classification and evaluation. In *ICDM*. 521–528.

Tsoumakas, G. and Katakis, I. 2007. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining 3,* 3, 1–13.

Uren, V. S. and Addis, T. R. 2002. How weak categorizers based upon different principles strengthen performance. *The Computer Journal 45,* 5, 511–524.

Vafaie, H. and Jong, K. A. D. 1994. Improving a rule induction system using genetic algorithms. In *Machine Learning IV: A Multistrategy Approach*, R. Michalski and G. Tecuci, Eds. Morgan-Kaufmann, 453–470.

van Rijsbergen, C. J. 1979. *Information Retrieval*, 2 ed. Butterworths, London.

Vilar, D., Castro, M. J., and Sanchis, E. 2004. Multi-label text classification using multinomial models. In *Proc. Espan a for Natural Language Processing (EsTAL'04).* Alicante, Spain.

Williams, N., Zander, S., and Armitage, G. 2006. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review 36,* 5 (Oct.), 5–16.

Witten, I. H. and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*, 2 ed. Morgan Kaufmann, San Francisco, CA.

Yang, Y. 1999. An evaluation of statistical approaches to text categorization. *Information Retrieval 1,* 1/2, 69–90.

Yang, Y. and Liu, X. 1999. A re-examination of text categorization methods. In *Proc. ACM SIGIR Conf. Research and Development in Information Retrieval.* Berkley, 42–49.

Yang, Y. and Pedersen, J. O. 1997. A comparative study on feature selection in text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, D. H. Fisher, Ed. Morgan Kaufmann Publishers, San Francisco, US, Nashville, US, 412–420.

Zhang, B. and Srihari, S. N. 2004. Fast k-nearest neighbor classification using cluster-based trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26,* 4, 525–528.

Zhang, M.-L. and Zhou, Z.-H. 2005. A k-nearest neighbor based algorithm for multi-label classification. In *The 1st IEEE Int'l Conf. on Granular Computing (GrC'05).* Vol. 2. Beijing, China, 718–721.

Zhang, M.-L. and Zhou, Z.-H. 2006. Multilabel neural networks with application to functional genomics and text categorization. *IEEE Trans. on Knowledge and Data Engineering*, 1338–1351.

Zhu, S., Ji, X., Xu, W., and Gong, Y. 2005. Multi-labelled classification using maximum entropy method. In *Proc. ACM SIGIR Conf. Research and Development in Information Retrieval*. 274–281.