Department of Computer Science

The University of Waikato
Te Whare Wānanga o Waikato

Hamilton, NewZealand

A Comparison of Multi-instance Learning Algorithms

Lin Dong

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science at The University of Waikato.

February 2006

Abstract

Motivated by various challenging real-world applications, such as drug activity prediction and image retrieval, multi-instance (MI) learning has attracted considerable interest in recent years. Compared with standard supervised learning, the MI learning task is more difficult as the label information of each training example is incomplete. Many MI algorithms have been proposed. Some of them are specifically designed for MI problems whereas others have been upgraded or adapted from standard single-instance learning algorithms. Most algorithms have been evaluated on only one or two benchmark datasets, and there is a lack of systematic comparisons of MI learning algorithms.

This thesis presents a comprehensive study of MI learning algorithms that aims to compare their performance and find a suitable way to properly address different MI problems. First, it briefly reviews the history of research on MI learning. Then it discusses five general classes of MI approaches that cover a total of 16 MI algorithms. After that, it presents empirical results for these algorithms that were obtained from 15 datasets which involve five different real-world application domains. Finally, some conclusions are drawn from these results: (1) applying suitable standard single-instance learners to MI problems can often generate the best result on the datasets that were tested, (2) algorithms exploiting the standard asymmetric MI assumption do not show significant advantages over approaches using the so-called collective assumption, and (3) different MI approaches are suitable for different application domains, and no MI algorithm works best on all MI problems.

Acknowledgments

First of all, I would like to thank my supervisor Dr. Eibe Frank who guided me through the whole research period and also allowed me to develop the project at my own pace. He always encouraged me and helped me reduce stress when I encountered difficulties or doubted myself. He taught me how to write a thesis and how to use Latex to write it up. Moreover, he proofread my thesis and commented on every aspect of this thesis. Without his encouragement and constant guidance, I could not have finished this thesis.

During the period of my study, the Department of Computer Science has provided me with much appreciated financial support and a graduate assistant position.

I would also like to thank all people working in the machine learning lab for the various kinds of help they have provided. I am grateful to Gabi Schmidberger for always being prepared to answer any questions I asked, Dale Fletcher for database-related help and proofreading my thesis, and Peter Reutemann and Richard Kirkby for general technical assistance.

I thank Soumya Ray, Stuard Andrews, Sally Goldman and Peter Reutemann for kindly providing their datasets.

I am so grateful to all friends I met in New Zealand, for their sincere friendship. Special thanks to Ann Li, Quan Qiu, Shaoqun Wu, Xiaofeng Yu and Wenlin Li for their constant help and encouragement.

Last, but not least, I thank my family for supporting and encouraging me in pursuing my interests.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Multi-instance (MI) learning has received much attention recently as the MI setting is well-suited to represent various complex real-world classification problems. Unlike standard supervised learning in which every training instance is explicitly labeled, in MI learning, the label information of every example is incomplete. Like standard supervised learning, the aim of MI algorithms is to generate a model based on training examples, so that the model can accurately make predictions on new examples. The prediction tasks for MI problems are more difficult than those for single-instance learning problems because the real cause of the class label is ambiguous. This thesis focuses on studying algorithms attempting to deal with MI classification problems.

The remainder of this chapter is organized as follows. Section 1.1 introduces some basic concepts of MI learning used throughout this thesis. Section 1.2 states the objectives of this thesis. Section 1.3 briefly reviews the history of MI learning and related work. Section 1.4 shows the structure of this thesis. Section 1.5 has some implementation notes.

1.1 Basic Concepts in Multi-instance Learning

The multi-instance concept was first formally proposed in (Dietterich, Lathrop & Lozano-Perez, 1997). It was originally motivated by a drug activity prediction problem. In this problem, a molecule can have several conformations (i.e. shapes) with different properties that result in the molecule being of "musk" or "non-musk" type. However, it is unknown which particular conformation is the cause of a molecule being of the "musk" type. The traditional single-instance setting can not represent this application problem properly as one molecule may have several alternative conformations. Therefore, Dietterich et al. (1997) proposed the multi-instance setting, in which each example is represented by a collection of single instances instead of a single instance.

In the MI setting, an example, which is called a "bag", consists of a collection of unlabeled instances. In this thesis, we only consider two-class classification problems, so each bag has a class label that is either 1 (for a positive example) or 0 (for a negative one). Each "instance" within a bag is described by a vector of features. However, the label of each individual instance within a bag is unobserved. In other words, the cause of the class label is not clearly known. For example, in the drug activity prediction problem, a molecule is represented as a bag that contains all possible conformations of this molecule, and every conformation of the molecule is represented as an unlabeled instance. In other words, a training example, which is either "musk" (positive) or "non-musk" (negative), is represented as a collection of unlabeled instances with only one associated bag label. Each instance is a fixed-length vector of nominal or numeric attribute values, just as in standard single-instance learning.

Dietterich et al. (1997) made an asymmetric MI assumption regarding the process that determines whether a bag label is positive or negative based on the (unknown) class labels of the instances in the bag. If and only if a bag contains at least one instance which is positive, the bag is labeled as positive. Otherwise, if all instances in a bag are negative, the bag is labeled as negative. This assumption has been regarded as the standard MI assumption and many MI algorithms that can be found in the literature follow this assumption.

However, several MI algorithms that have been proposed more recently, especially methods aimed at upgrading single-instance learners to deal with MI data, discarded the standard MI assumption and instead use a so-called "collective assumption" (Xu, 2003). The collective assumption is symmetric. It assumes all individual instances within a bag contribute equally and independently to the bag's class label.

This thesis discusses MI learning algorithms. To avoid confusion it refers to standard supervised learning algorithms as "single-instance learning algorithms". In the single-instance setting, each training instance is assigned an explicit class label whereas in the multi-instance setting, the class label is assigned to a bag of instances. Therefore, the task of accurately classifying or predicting is more difficult in MI learning.

1.2 Thesis Objectives

In recent years, many MI algorithms have been proposed in the literature. However, in most cases, the evaluation of these algorithms was based on very few MI datasets (e.g. the two Musk benchmark datasets) and the proposed algorithms were often compared with very few other approaches (e.g. an early MI learning method called Diverse Density). The aim of this thesis is to present a comprehensive study of MI methods. To this end extensive experiments on a collection of real-world datasets were performed in an attempt to address some specific questions regarding MI problems: such as how well standard single-instance learners work on MI problems, which kind of MI assumption is most suitable for real-world dataset, and which MI algorithm is most suitable for a specific application domain. More precisely, there were three objectives for the work presented in this thesis:

1. To integrate all MI classifiers from the MILK system for multi-instance learning (Xu, 2003) into the WEKA workbench and to implement more well-known MI algorithms;
2. To evaluate the performance of various kinds of MI approaches on a wide range of real-world datasets;
3. To perform an in-depth study of how well different kinds of MI approaches work on MI problems.

A major part of this thesis is also a review of the MI learning algorithms that were evaluated.

1.3 Background and Related Work

In this section, we briefly review the history of research on MI learning. In the early stage of development, the MI algorithms that were created were often specially designed for the MI setting. In the later stages, researchers showed more interest in upgrading or directly applying standard single-instance learning algorithms to MI problems.

In the area of MI learning, the article that introduced the axis-parallel rectangles (APRs) algorithm (Dietterich, Lathrop & Lozano-Perez, 1997), can be regarded as the first paper to formally introduce the multi-instance framework and it proposed the first MI algorithm specifically designed to learn from MI examples. The APRs method assumes that the classifier can be represented as a hyper-rectangle that includes at least one instance of every positive bag and does not include any instances from negative bags. The MI setting for the drug activity prediction problem was first introduced in this paper, which resulted in the two versions of the Musk benchmark datasets. Soon after, a related algorithm based on the PAC theory was introduced under the MI framework (Auer, Long & Srinivasan, 1997), and then a more practical algorithm called *MULTINST* was presented in (Auer, 1997). The *MULTINST* algorithm assumes that all instances from all bags are generated independently and the algorithm is based on simple statistics of the bags and designed to efficiently learn APR concepts.

In 1998, Maron et al. proposed a new MI algorithm called Diverse Density (*DD*) and this algorithm is commonly used for bench-marking other MI algorithms. It assumes that there is a concept point in the feature space that is close to all positive bags and far away from all negative bags. The *DD* algorithm is very famous and has been cited in almost every paper on MI learning. A few years later, the *DD* algorithm was extended further by combining it with the Expectation-Maximization (EM) algorithm, resulting in the *EMDD* algorithm (Zhang & Goldman, 2002). *DD* and its extension *EMDD* have been used on several MI problems, namely drug activity prediction, stock selection, natural scene classification and image retrieval. These algorithms will be discussed in more detail in Section 2.1.

Since 2000, much effort has been targeted at adapting single-instance learning algorithms to MI problems. An MI version of the C4.5 decision tree learner, called *RELIC*, was presented in (Ruffo, 2000). Wang and Zucker (2000) explored a lazy learning approach to MI learning. They adapted the k nearest neighbor algorithm to MI problems by using the *Hausdorff Distance* for measuring the distance between sets of point. Two variants of this approach, *Citation KNN* and *Bayesian KNN*, were proposed in (Wang & Zucker, 2000). They will be discussed in more detail in Section 2.3. Also in 2000, Ramon and Raedt applied the Neural Network learning technique on MI learning (Ramon & Raedt, 2000). In 2001, an MI version of a decision rule learner called *RIPPER* was proposed in (Chevaleyre & Zucker, 2001). In 2002, several methods attempting to adapt the support vector machine (SVM) method to MI learning were investigated. Gärtner et al. (2002) proposed a kernel-based method that uses an MI-kernel at the bag level. Andrews et al. (2002) proposed two methods to utilize the standard SVM to solve MI problems. One is to identify the unobserved class label for each individual instance using a standard SVM. The other is to generalize each bag by searching for the "most positive" instance and the "least negative" instance for positive and negative bags respectively. SVM-based methods will be discussed in more detail in Section 2.2.

In 2003, Frank and Xu introduced a simple wrapper for applying standard single-instance learner directly to MI problems. In the *Wrapper* approach (Frank & Xu, 2003), the standard asymmetric MI assumption is explicitly discarded and the collective assumption is used, which treats the positive and negative bags in a symmetric way. Moreover, Frank and Xu introduced a new weighting scheme to properly treat instances from different bags differently. This method will be discussed in more detail in Section 2.4. Later, Xu and Frank (2004) also proposed a method to upgrade linear logistic regression and boosting to MI problems, which also used the collective assumption. Ensemble methods have also been suggested to combine MI learners in order to achieve a better performance, such as the bagging approach (Zhou & Zhang, 2003) and the boosting approach (Auer & Ortner, 2004).

Apart from MI learning for classification problems, efforts have also been made on developing MI methods dealing with real-valued outputs. The MI algorithms *Citation KNN* and *DD* were extended for the real-valued data setting in (Amar, Dooly, Goldman & Zhang, 2001). An MI regression algorithm was also proposed in (Ray & Page, 2001). As MI learning was originally motivated by two class problems, this thesis focuses on investigating binary class problems and real-valued prediction problems are beyond its scope.

Many MI algorithms have been proposed and many MI application domains have been investigated, but the scope of existing studies of MI learning is very limited. Researchers often pay a lot of attention to the performance of their newly proposed algorithm on the traditional drug activity prediction task and a few particular application domains they are interested in. The result is that very limited comparisons between different MI algorithms and different application domains are performed. Many algorithms appear to work very well on the two versions of the classic Musk dataset for drug activity prediction. However, a perfect solution for the Musk datasets should obviously not be the final goal of MI learning.

The only exception to this is the very recent work by Ray and Craven (2005). They performed an empirical study of the relationship between standard supervised learning and MI learning on a non-trivial collection of datasets. They attempted to answer questions such as, how well do standard supervised learners do on MI data, and is there any single MI algorithm well suited to every MI domain. In this study (Ray & Craven, 2005), a number of MI learning algorithms including *DD*, *Logistic Regression*, *SVM* and *FOIL* were empirically compared with their corresponding standard single-instance learning counterpart. The evaluation was based on a wide range of MI problems that have previously been considered in the MI literature. The application areas included drug activity prediction (the Musk datasets), content based image retrieval, identification of Trx proteins and text categorization. The conclusion was that some MI algorithms, such as *DD* and *Logistic Regression*, are always superior to their single-instance counterparts but that single-instance learning algorithms learn accurate models in many MI domains and sometimes are the best algorithms (Ray & Craven, 2005). The empirical results also show that different MI algorithms are appropriate for different MI problems and that no single MI algorithm was well-suited to every MI domain that was tested. Compared with the work by Ray and Craven (2005), this thesis will provide a more comprehensive study of MI algorithms. A total of 16 MI algorithms will be investigated on a total of 15 real-world datasets.

1.4 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 presents the MI algorithms that have been investigated, divided into five different kinds of approaches: probabilistic approaches, support vector machine approaches, distance-based approaches, approaches based on applying single-instance learners and two-level distribution approaches. For each MI approach, we describe its main idea and introduce several methods that instantiate it. A total of 16 individual MI algorithms that have been implemented in WEKA (Witten & Frank, 2005) are discussed in this chapter.

Chapter 3 empirically evaluates the MI algorithms described in this thesis on a wide range of real-world datasets. The MI application domains considered were collected from previous projects involving MI learning, including drug activity prediction, image retrieval, protein identification, the East-West challenge and text categorization. Extensive experiments were performed on these datasets. The empirical results of these experiments are compared and discussed in this chapter. At the end of Chapter 3, an attempt is made to identify suitable MI approaches for different application domains.

Chapter 4 summarizes the findings of this thesis and compares them with the findings from previous related work (Ray & Craven, 2005).

1.5 Implementation Notes

For the experiments in this thesis all the classifiers from the MILK system (Xu, 2003) were integrated into WEKA and four more MI algorithms were implemented. More precisely, the work included:

1. Integrating all the MI classifiers in MILK system into the *weka.classifiers.mi* package.
2. Implementing *MIEMDD*, *MISVM*, *MIOptimalBall*, *MISMO*, and adding a third transformation method, *minimax*, to the *SimpleMI* classifier.
3. Collecting the MI datasets used in this study and running the experiments using WEKA.

In the following is a full list of all MI classifiers in the *weka.classifiers.mi* package, which will be included in the next development release of WEKA:

- MIDD: Diverse Density classifier
- MIEMDD: Expectation-Maximization Diverse Density
- MDD: Diverse Density with collective assumption
- MILR: Logistic Regression with standard MI assumption
- MILRARITH: Logistic Regression with collective assumption (arithmetic mean)
- MILRGEOM: Logistic Regression with collective assumption (geometric mean)
- MISMO: SMO algorithm for building SVMs for MI data using MI-kernels
- MISVM: Maximum Pattern Margin Formulation
- CitationKNN: a lazy learning approach for MI problems
- MINND: Nearest Distribution classifier
- MIOptimalBall: Optimal Ball classifier

- MIWrapper: a method for applying single-instance learners to MI problems
- SimpleMI: applying single-instance learners to MI data by summarization
- MIBoost: Boosting for MI data
- TLD: Two-level Distribution classifier
- TLDSimple: a simplified version of the Two-level Distribution classifier

These methods will all be discussed in Chapter 2.

Chapter 2
Algorithms

In recent years, more and more algorithms for multi-instance (MI) learning have been proposed by researchers. Some of the MI algorithms are particularly designed to learn MI concepts whereas some are adapted from standard single-instance learning algorithms. In this chapter, we are going to explore a total of sixteen MI algorithms. According to the underlying ideas of these MI algorithms, they are divided into five groups, which result in five separate sections. They are probabilistic approaches, support vector machine approaches, distance-based approaches, methods that apply single-instance learners to MI problems and two-level distribution approaches.

2.1 Probabilistic Approaches

In this section, we review methods for MI learning that are based on probabilistic models. The basic idea is to define a suitable probability function or probability density function for the MI problem and estimate the parameters of the function based on the training data, using a statistical criterion, i.e. maximum likelihood estimation, to find the most likely values of the parameters. At testing time, we determine the class label of a new bag by assigning the class label with maximum probability.

We first look at a well-known MI learning approach called Diverse Density (*DD*) (Maron, 1998; Maron & Lozano-Perez, 1998), discuss variants of it and then discuss the logistic regression approach as applied to MI learning (Xu & Frank, 2004). More specifically, we discuss the standard *DD* algorithm, an extension of *DD* that combines the general EM approach with the *DD* model, called Expectation-Maximization Diversity Density (*EMDD*) (Zhang & Goldman, 2002), and another version of *DD* that uses the so-called "collective" assumption instead of the "standard" MI assumption. We then discuss how to apply the general logistic regression approach to solve MI problems and cover three different kinds of logistic regression models based on different MI assumptions.

2.1.1 Diverse Density

The Diverse Density (DD) approach was the first probabilistic model developed for MI learning. It was originally proposed by Maron and Lozano-Perez in 1998 (Maron, 1998; Maron & Lozano-Perez, 1998). They define Diverse Density at a point to be "a measure of how many different positive bags have instances near that point, and how far the negative instances are from that point" (Maron & Lozano-Perez, 1998). The basic idea of the DD approach is to find a point in the feature space with high Diverse Density. In other words, a candidate point is one that is close to as many different positive bags as possible and at the same time as far away from the negative bags as possible.

Maron and Lozano-Perez derived a probabilistic measure of DD at a single point, which can be regarded as the probability of this point representing the "true concept" (i.e. the target point). The DD method assumes that the "true concept" can be retrieved by maximizing the probabilistic measure of DD through the whole feature space. Let us denote the positive bags in the training data as B^+, the negative bags as B^- and the jth point in the ith bag as B_{ij}. Therefore, B^+_{ij} represents the jth point in the ith bag, whose bag label is positive. The probabilistic measure of DD can then be written as (Maron & Lozano-Perez, 1998):

$$DD(x) = Pr(x = t | B^+_1, \dots, B^+_n, B^-_1, \dots, B^-_m). \tag{2.1}$$

Here, x denotes a single point in the feature space and t denotes the point that is the "true concept". The goal is to maximize this DD measure. Applying Bayes rule and assuming a uniform prior over concept locations, maximizing the DD function is equivalent to maximizing the following likelihood function:

$$\arg\max_x Pr(B^+_1, \dots, B^+_n, B^-_1, \dots, B^-_m | x = t),$$

because using Bayes rule, Equation 2.1 can be written as

$$DD(x) = \frac{Pr(B^+_1, \dots, B^+_n, B^-_1, \dots, B^-_m | x = t) Pr(x = t)}{Pr(B^+_1, \dots, B^+_n, B^-_1, \dots, B^-_m)}.$$

Here, $Pr(x = t)$ is assumed to be a uniform prior that is constant. $Pr(B^+_1, \dots, B^+_n, B^-_1, \dots, B^-_m)$ is the probability of producing the data which is also constant and can be regarded as a normalizing term that need not be calculated explicitly (Maron, 1998). Therefore, the likelihood is the only term left to compute.

Assuming further that all the bags are conditionally independent given the target concept, the following equivalent expression can be achieved:

$$\arg\max_x \prod_i Pr(B_i^+|x=t) \prod_i Pr(B_i^-|x=t).$$

Finally, applying Bayes rule and assuming a uniform prior over concept locations again, Maron and Lozano-Perez derived a general definition of maximum *DD* as follows (Maron & Lozano-Perez, 1998):

$$\arg\max_x \prod_i Pr(x=t|B_i^+) \prod_i Pr(x=t|B_i^-). \tag{2.2}$$

DD assumes that the bags are conditionally independent, so each bag in the feature space can be viewed as a Bernoulli trial and a single point x being the "true concept" as an event (Maron & Lozano-Perez, 1998). This results in the bag-level probability product shown in Equation 2.2 . This is a bag-level expression. The individual instances in the bags are not mentioned. For a positive bag and a negative bag, a single point x being the "true concept" has different implications. For a positive bag, x being the "true concept" implies that x should be close to the bag. Therefore, the first product term in the equation actually measures the closeness between the target point and the positive bags. On the other hand, for a negative bag, x being the "true concept" implies that x should be far away from this bag. Therefore, the second product term in the equation actually measures how far the target point is away from the negative bags.

In order to compute the two product terms in Equation 2.2, we need to consider how a single instance in a bag influences the bag-level class probability. Two different models have been proposed by Maron (1998) to represent the bag-level construction. One is called the "noisy-or model" and the other is called the "most-likely-cause model".

- Noisy-or model

  The noisy-or model was first introduced by Pearl (1988) in the context of Bayesian networks. The word *noisy* means each cause influences the result with some probability and it is not a deterministic cause. The word *or* indicates the method used to combine all these independent causes. It is also suitable for the MI case. In the noisy-or model, it is assumed that the event can only happen if at least one of the causations occurred and the probability of any cause failing to trigger the event is independent (Maron, 1998). If we assume each single instance within a bag has its own label but it is not clear whether it is positive or negative, and each instance can be viewed as an independent cause of a positive bag label that influences the label with some probability, then we arrive at the noisy-or model. The standard MI assumption implies the fact that the boolean bag label (either negative 0 or positive 1) is the result of a logic OR of all the single-instance labels within that bag. Given the MI assumption, the probability that a bag is positive is equivalent to the probability that not all individual instances within that bag are negative. On the other hand, the probability that a bag is negative is equivalent to the probability that all individual instances within that bag are negative.

The noisy-or model can be expressed as (Maron, 1998):

$$Pr(x = t|B_i^+) = 1 - \prod_j (1 - Pr(x = t|B_{ij}^+))$$

and

$$Pr(x = t|B_i^-) = \prod_j (1 - Pr(x = t|B_{ij}^-)).$$

Here, the probability $Pr(x = t|B_{ij})$ is a measure of the closeness between a point B_{ij} and the target point x (discussed below). In other words, it is the probability of a single instance being positive. Therefore, $1 - Pr(x = t|B_{ij})$ refers to the probability of a single instance's label being negative. Again, as we have assumed each instance within a bag is an independent cause of the bag label, the processing of instances within a bag can be considered as Bernoulli trials with different probability. This results in the expression $\prod_j (1 - Pr(x = t|B_{ij}))$ in the two above equations.

- Most-likely-cause model

  Unlike the noisy-or model, the most-likely-cause model only considers a single representative instance of each bag. The model picks the instance from each bag that is most likely to be responsible for the bag label. More specifically, the representative instance of a bag is the one which has the highest probability to be positive in that bag.

It is obvious that the most-likely-cause model is different from the noisy-or model in that it only considers the "key" point within each bag. This point represents a single Bernoulli trial within each bag and therefore the most-likely-cause model does not involve a product expression. The most-likely-cause can be expressed as (Maron, 1998) :

$$Pr(x = t|B_i^+) = \arg\max_j Pr(x = t|B_{ij}^+)$$

and

$$Pr(x = t|B_i^-) = 1 - \arg\max_j Pr(x = t|B_{ij}^-).$$

The most-likely-cause model also conforms to the standard MI assumption. In the two-class case, we assume the threshold for making a positive classification is set to 0.5. Then the most-likely-cause model implies that in a positive bag, the instance with the highest probability to be positive has a probability greater than or equal to 0.5 to be positive. This is equivalent to assuming that at least one instance within a positive bag is positive. Similarly, in a negative bag, the instance with the highest probability to be positive should have a probability of being positive of less than 0.5. This is equivalent to all instances within a negative bag being negative.

Now the last question is how to model the instance-level probability $Pr(x = t|B_{ij})$. An instance-based probability estimator is required. Maron (1998) proposed to model the probability related to the distance between the target point x and a single-instance point in the feature space. So the *DD* algorithm uses the Euclidean distance metric to measure the distance and the probability can then be estimated by a Gaussian-like distribution whose center is the target point x. If the single-instance point is close to the the target point, the probability $Pr(x = t|B_{ij})$ is high. Otherwise, if it is far away from point x, the probability is low. Furthermore, it is necessary to find the best scaling for each individual feature because each feature's influence on the bag label is quite different in many cases. Some features are very important while some are even irrelevant. The generative model for the instance-level probability can be expressed as follows (Maron, 1998):

$$Pr(x = t|B_{ij}) = \exp(-\sum_k s_k^2(B_{ijk} - x_k)^2). \tag{2.3}$$

Here, k denotes the kth feature, B_{ijk} denotes the value of the kth feature of the point B_{ij}. x is the target point vector and s is a scaling vector for the features. So the target concept actually consists of two values for each feature. Formally, suppose each point has k features. Then the target concept can be expressed as $x = \{x_1, ..., x_k, s_1, ..., s_k\}$.

One problem of searching for the maximum *DD* measure through the whole feature space is that we need to ensure the point being found is a global maximum value not just a local maximum. The optimization algorithm may just find a local maximum, and different starting points may result in different target points. Hence, it is important to try some sensible starting points. The *DD* algorithm uses gradient ascent with multiple starting points (eg. starting from each point from each positive bag) to find the point that maximizes the diverse density function in the hope that the global maximum can be found. However, trying all points in all positive bags as suggested in (Maron, 1998) can be extremely time consuming, especially for datasets with large bags. In order to make the *DD* algorithm run faster, the implementation of *DD* with the noisy-or model (Xu, 2003) used for the experiments in this thesis tries all the points in the positive bags with the largest size. This is not necessarily detrimental. According to (Xu, 2003), this strategy gives even a higher accuracy on the Musk1 dataset than the strategy of trying every point in all positive bags.

In summary, *DD* (with either the noisy-or model or the most-likely-cause model) matches the standard MI assumption well. It has been shown to produce good results in solving MI problems (Maron, 1998; Maron & Lozano-Perez, 1998). Apart from this, the most-likely-cause and the noisy-or models provide a general method for converting the multi-instance setting to a single-instance setting.

2.1.2 Expectation-Maximization Diverse Density

Expectation-Maximization Diversity Density (*EMDD*) is an extension of the *DD* algorithm (Zhang & Goldman, 2002). It combines the Expectation-Maximization algorithm (EM) with the *DD* algorithm used with the most-likely-cause model. As we have seen, the most-likely-cause model involves the "maximum" function and this causes a non-differentiable optimization problem in the *DD* algorithm with the most-likely-cause model. Maron (1998) uses a "softmax" approximation for the "maximum" function to solve the problem. However, this greatly increases the computing time. *EMDD* aims to avoid this non-differentiable problem by solving the "maximum" function in the expectation step (E-step) of the EM algorithm. Moreover, according to (Zhang & Goldman, 2002) the EM process may help to avoid getting trapped in a local maximum of the likelihood because the EM process iteratively updates the previous target point.

Let us discuss how *EMDD* works. Figure 2.1 shows the *EMDD* algorithm. Like the general EM approach, *EMDD* starts with some initial hypothesis as the guess of the target point x. With each different initial hypothesis, it repeats the E-step and M-step combined with the *DD* algorithm to iteratively search for the hypothesis with the maximum likelihood. In the implementation of *EMDD* written for the experiments in this thesis, each point in three randomly picked positive bags is tried as the initial hypothesis. For the expectation step (E-step), we use the hypothesis x as a base to pick a single instance from each bag: the one that is the most likely to be responsible for each bag label. Thus the E-step converts the multi-instance setting into a single-instance setting. As a result, a set of instances in which each single instance represents one bag has been extracted. For the maximization step (M-step), we apply the *DD* algorithm with the most-likely-cause model to search for a new hypothesis x' that maximizes the *DD* likelihood function and record its corresponding *DD* measure value. Then, like in the normal EM process, we update the hypothesis to the new point x' and repeat the above two steps for more iterations until the algorithm has converged.

The convergence test is performed before each execution of the E-step. If the *DD* measure of the current iteration is less than or equal to that of the previous iteration, we regard it as convergence. Note that in Figure 2.1, the negative logarithm of the *DD* measure ($NLDD$) has been considered instead. The reason for that is that our optimization procedure is formulated in terms of minimization rather than maximization, and we are looking for a minimum value. The stopping condition turns out to be that the $NLDD$ value of the current iteration ($nldd_1$) is greater than or equal to the $NLDD$ value obtained from the previous iteration ($nldd_0$). Moreover, in order to reduce the training time, we set the maxmimum number of iterations to be 10 as suggested in (Zhang & Goldman, 2002). Usually, the $NLDD$ value decreases dramatically after the first several iterations and then begins to flatten out.

However, the *EMDD* algorithm described in the original paper (Zhang & Goldman, 2002) had some problem in the selection of the best hypothesis. Several later papers (Andrews, Tsochantaridis & Hofmann, 2002; Xu, 2003) have also pointed out the

```
Build Classifier:
Normalize or standardize the training data (determined by user);
Let $minError = +\infty$;
Randomly pick 3 positive bags from the training data;
For each instance $I_j$ from each selected bag
        Initialize the starting hypothesis $x$:
            Let $x = \{x_1, ..., x_n, s_1, ..., s_n\}$;
            For each dimension $k = 1, ..., n$
                $x_k = I_k$;    $s_k = 1.0$;
        $nldd_0 = +\infty$;
        $nldd_1 =$ a large value smaller than $nldd_0$;
        $iterationCount = 0$;

        while ( $nldd_1 < nldd_0$ or $iterationCount < 10$ )
            $iterationCount++$;
            E-step:
            For each bag $B_i$ in the feature space
                Find the representative instance $p_i^*$ given hypothesis $x$:
                $p_i^* = argmax_{B_{ij}} Pr(x = t|B_{ij})$
                    where $Pr(x = t|B_{ij}) = \exp[-\sum_{k=1}^{n}(s_k^2(B_{ijk} - x_k)^2)]$;
            M-step:
            Find the target point $x'$ with the maximum Diversity Density given the $p_i^*$:
            $x' = \arg\max_x(\prod_i Pr(x = t|B_i^+) \prod_i Pr(x = t|B_i^-))$
                where $Pr(x = t|B_i^+) = \exp[-\sum_{k=1}^{n}(s_k^2(p_{ik}^* - x_k)^2)]$
                and $Pr(x = t|B_i^-) = 1 - \exp[-\sum_{k=1}^{n}(s_k^2(p_{ik}^* - x_k)^2)]$;
            $nldd_0 = nldd_1$;
            $nldd_1 = NLDD(x')$;
            Update the hypothesis as:
                $pre\_x = x$;
                $x = x'$;

        Let $error = 0$;
        If $nldd_1 > nldd_0$
            Let $h = pre\_x$;
        Else
            Let $h = x$;
        Evaluate the hypothesis $h$ on the training data and count the $error$;
        Keep track of the best hypothesis that has the minimum number of errors:
            If $error < minError$
                $minError = error$;
                Update the best hypothesis: $X = h$;

Classify:
Normalize or standardize the unknown bag data $B_i$;
Initialize $distribution$;
For each instance within the bag $B_i$
     calculate the likelihood using the best hypothesis $X$, and find the maximum value:
     $distribution[1] = \max_{B_{ij}}(\exp[-\sum_{k=1}^{n}(S_k^2(B_{ijk} - X_k)^2)])$;
     $distribution[0] = 1 - distribution[1]$;
return $distribution$;
```

Figure 2.1: *EM-DD* algorithm, adapted from (Zhang & Goldman, 2002)

problem. In the *EMDD* algorithm, the hypothesis candidates are all the hypothesis obtained from each completed *EM* iteration with the different initial starting points. In the paper (Zhang & Goldman, 2002), the final model selection is made according to the error rate on the testing data instead of the training data. Therefore the high accuracy (over 96%) that was obtained using 10-fold cross-validation on the musk benchmark datasets, which will be discussed in Section 3.1.1, is not convincing. In our implementation, we correct this mistake by choosing the best hypothesis based on the training data.

EMDD is suitable for datasets with large bags because it converts each multi-instance bag into a single instance before *DD* is run. Thus this application of the most-likely-cause model can, theoretically, greatly decrease the time cost for the optimization process.

2.1.3 Diverse Density with Collective Assumption

The standard *DD* algorithm and its *EMDD* variant both adopt the standard MI assumption. However, another kind of assumption has been introduced for MI learning by Xu (2003), that resulted in high accuracy for some practical problems such as drug activity prediction. This assumption is called the "collective assumption". The collective assumption regards the class label of a bag as a collective property that is related to all the instances within that bag. Compared with the standard MI assumption, which implicitly assumes that there is a "key" positive instance within a positive bag, the collective assumption equally considers every instance's contribution to the bag label as being equally important.

Based on this new assumption, another modified version of *DD* has been developed (Xu, 2003) by discarding the standard MI assumption and instead applying the collective assumption. We assume that all instances within a bag contribute independently and equally to a bag's class label. Under this new assumption, we now average the probability $Pr(x = t|B_{ij})$ associated with each single instance within a bag. Thus we can express the bag-level probability as follows:

$$Pr(x = t|B_i^+) = \frac{1}{N_i}\sum_j Pr(x = t|B_{ij}^+)$$

for a positive bag B_i^+, and

$$Pr(x = t|B_i^-) = \frac{1}{N_i}\sum_j (1 - Pr(x = t|B_{ij}^-))$$

for a negative bag B_i^-. Here, N_i denotes the number of instances within the ith bag. This model is learned using the same algorithm as the standard *DD* model.

2.1.4 Logistic Regression Model

In standard single-instance supervised learning, the linear logistic regression algorithm fits a linear model to the log-odds of the class probabilities. At classification time, an unknown instance is assigned to the class whose corresponding linear function value is maximum among all the classes. For a two-class problem where $y = 1$ denotes the positive label and $y = 0$ denotes the negative label, the two resulting instance-level functions are

$$Pr(y = 1|x) = \frac{1}{1 + \exp(-\beta x)}$$

and

$$Pr(y = 0|x) = \frac{1}{1 + \exp(\beta x)}.$$

The weight vector β is found by fitting the model to the training data. Unlike linear regression, which determines the weight vector by minimizing the squared error, logistic regression determines it by maximizing the log-likelihood function. The likelihood is the product of the probability for each training instance given the model. Usually, to avoid arithmetic underflows, the logarithm of the likelihood is used, in which case the product of probabilities turns into a sum of logarithms of probabilities. The greater the log-likelihood, the better the model fits to the data.

However, for MI problems, the standard logistic regression model cannot be applied directly as the class label for each single instance within a bag is not known. An indirect estimate of the logistic model has been proposed in (Xu & Frank, 2004). They extend the standard instance-based logistic regression model to be a bag-level model under specific assumptions. The assumptions indicate how the instance-level class probabilities are combined to form the bag-level probability so that the actual class label for each instance is not required. The details of these are discussed below.

The bag-level log-likelihood function can then be expressed as follows, in the same form as in normal supervised learning:

$$LL = \sum_{i}^{N} [y_i \log Pr(y = 1|b) + (1 - y_i) \log Pr(y = 0|b)]. \tag{2.4}$$

Here, N denotes the total number of bags. y_i denotes the ith bag label (either 0 or 1). b denotes a single bag. Now the question is how to generate the two bag-level probabilities $Pr(y = 1|b)$ and $Pr(y = 0|b)$. Like in standard single-instance supervised learning, we model the class probability for each individual instance within a positive bag and a negative bag as $Pr(y = 1|x) = \frac{1}{1+\exp(-\beta x)}$ and $Pr(y = 0|x) = \frac{1}{1+\exp(\beta x)}$ respectively. Different assumptions regarding the combination of these instance-level probabilities to bag-level probabilities lead to different models. Here, we briefly illustrate three different kinds of logistic regression models. The first two models have been presented in (Xu & Frank, 2004) and both discard the standard MI assumption and use the collective assumption (Xu, 2003) instead. To make a comparison, the third model applies the standard MI assumption.

- Multi-instance logistic regression with arithmetic mean model

  The collective assumption assumes that each individual instance within a bag contributes independently and equally to the bag label. Under this assumption, it is natural to look at the average instance-level probabilities within a bag. For each bag, the model simply calculates the arithmetic mean of the probabilities for each instance to construct the conditional probability of the bag-level class label. This can be expressed as follows:

$$Pr(y = 1|b) = \frac{1}{n} \sum_{i}^{n} \frac{1}{1 + \exp(-\beta x_i)}$$

$$Pr(y = 0|b) = \frac{1}{n} \sum_{i}^{n} \frac{1}{1 + \exp(\beta x_i)}.$$

  Here, n denotes the number of instances within the bag b and x_i denotes the ith instance in the bag.

- Multi-instance logistic regression with geometric mean model

  This model uses the same assumption as the previous one. The difference is that this model actually uses the geometric mean to implement the collective assumption instead of the previous arithmetic mean. Xu and Frank (2004) derive this bag-level model by estimating the log-odds function instead of directly estimating the probability function as follows:

$$\log \frac{Pr(y=1|b)}{Pr(y=0|b)} = \frac{1}{n}\sum_{i=1}^{n} \log \frac{Pr(y=1|x_i)}{Pr(y=0|x_i)}.$$

The bag-level log-odds function is computed as the average of the instance-level log-odds functions. When this is done, $Pr(y = 1|b)$ and $Pr(y = 0|b)$ can then be written as (Xu & Frank, 2004)

$$Pr(y=1|b) = \frac{[\prod_i^n Pr(y=1|x_i)]^{\frac{1}{n}}}{[\prod_i^n Pr(y=1|x_i)]^{\frac{1}{n}} + [\prod_i^n Pr(y=0|x_i)]^{\frac{1}{n}}}$$

$$Pr(y=0|b) = \frac{[\prod_i^n Pr(y=0|x_i)]^{\frac{1}{n}}}{[\prod_i^n Pr(y=1|x_i)]^{\frac{1}{n}} + [\prod_i^n Pr(y=0|x_i)]^{\frac{1}{n}}}.$$

This is where the geometric mean comes from. Modeling each single-instance within a bag as $Pr(y = 1|x) = \frac{1}{1+\exp(-\beta x)}$ and $Pr(y = 0|x) = \frac{1}{1+\exp(\beta x)}$, we can finally obtain the bag-level probability functions as (Xu & Frank, 2004):

$$Pr(y=1|b) = \frac{\exp(\frac{1}{n}\beta\sum_i x_i)}{1+\exp(\frac{1}{n}\beta\sum_i x_i)}$$

$$Pr(y=0|b) = \frac{1}{1+\exp(\frac{1}{n}\beta\sum_i x_i)}.$$

It is easy to see that these two expressions can be rewritten as

$$Pr(y=1|b) = \frac{\exp(\beta\bar{x})}{1+\exp(\beta\bar{x})}$$

$$Pr(y=0|b) = \frac{1}{1+\exp(\beta\bar{x})},$$

where $\bar{x}$ denotes the mean point of a bag. It shows that this model is equivalent to converting the multi-instance data to single-instance data by averaging all the instances within a bag and then directly applying the standard single-instance linear logistic regression model to the converted dataset. This conversion can be performed by simply extracting the mean of each bag as a representative of the bag.

- Multi-instance logistic regression with standard MI assumption

The standard MI assumption requires that there is at least one positive instance within a positive bag and no positive instance within a negative bag. Like in one version of the *DD* model, which also adopts the MI assumption, we can use the same noisy-or model here to represent this assumption. The model can be expressed as follows:

$$Pr(y=1|b) = 1 - \prod_i^n (1 - Pr(y=1|x_i)) = 1 - \prod_i^n \frac{1}{1+\exp(\beta x_i)}$$

$$Pr(y=0|b) = \prod_i^n Pr(y=0|x_i) = \prod_i^n \frac{1}{1+\exp(\beta x_i)}.$$

For these three kinds of models based on the three different assumptions, the general form of the bag-level log-likelihood function is the same (see Equation 2.4). The weight parameter of the model β can be found by maximizing the bag-level log-likelihood function based on the training data. Using the standard optimization process, the parameters are found iteratively and the iterations stop when the improvement from one step to the next is small enough. A quasi-Newton optimization procedure with BFGS updates is used in the implementation (Xu, 2003).

2.2 Support Vector Machine Approaches

The support vector machine (SVM) is a supervised learning algorithm developed by Vapnik (1995). The SVM approach uses linear models to achieve non-linear class boundaries. To this end it maps the data into a new instance space (a high-dimensional feature space) by a non-linear mapping so that linear models for the transformed data correspond to non-linear models in the original instance space. The non-linear mapping can be performed efficiently using a so-called "kernel" function.

In this section, we are going to study how to apply the supervised SVM approach when solving MI problems. Two different kinds of solutions will be discussed. The first solution aims to derive bag-level MI kernels (Gärtner, Flach, Kowalczyk & Smola, 2002). The second solution attempts to identify the unobserved class label for each single-instance by using the standard SVM solution (Andrews, Tsochantaridis & Hofmann, 2002).

2.2.1 Multi-instance Kernels

In standard single-instance SVMs, training data is given as a set of instance-label pairs (x_i, y_i), and they are non-linearly mapped to a high-dimensional space F. The SVM algorithm aims to find a maximum margin hyperplane in F that can linearly separate two classes. As it can be shown that all information required to find this linear separation consists of inner products of each pair of the original data points, the mapping process to the high-dimensional space does not actually have to be performed explicitly. Instead, a kernel function, which plays the role of the dot product, needs to be defined and this function is evaluated for each pair of original data points. If the kernel function K is defined, the decision boundary can be expressed as:

$$\sum_i \alpha_i y_i K(x_i, x) + b = 0. \tag{2.5}$$

Here, x_i is the training data and y_i is the corresponding class label. α_i and b are the parameters that need to be determined. All points x for which this equation holds are on the decision boundary.

Two commonly used kernel functions are:

- polynomial: $K(x_i, x_j) = (x_i \cdot x_j)^d$
- radial basic function (RBF): $K(x_i, x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2), \quad \gamma > 0.$

When testing, an unknown instance x' can be classified by the obtained SVM function:

$$f(x) = \sum_i \alpha_i K(x_i, x) + b. \tag{2.6}$$

where x_i is the training data point. The point x_i is called a "support vector" if the corresponding parameter α_i is non-zero. If the resulting value $f(x')$ is greater than 0, the unknown instance is classified as positive. Otherwise, it is classified as negative.

For MI problems, the supervised SVM algorithm can not be applied directly because the kernel function requires the explicit instance-label pairs (x_i, y_i). However, this problem can be solved by using a bag-level multi-instance kernel function (MI kernel) (Gärtner, Flach, Kowalczyk & Smola, 2002). Once the bag-level kernel is defined, we can make use of bag-label pairs (B_i, Y_i) instead of instance-label pairs (x_i, y_i) in Equation 2.5 so that the SVM approach can be easily applied to MI problems.

The MI kernel is derived from prior work on kernels for discrete spaces (Haussler, 1999; Gärtner, 2000). There, a kernel on sets is defined as follows:

$$K_{\text{set}}(X, X') := \sum_{x \in X, x' \in X'} K_{\chi}(x, x')$$

where $X, X' \subset \chi$ and K is a kernel on χ.

Based on this set kernel definition, a kernel that can separate MI concepts is defined as follows (Gärtner, Flach, Kowalczyk & Smola, 2002):

$$K_{\text{MI}}(X, X') = \sum_{x \in X, x' \in X'} K_I^p(x, x') \qquad (2.7)$$

where $p \in \mathcal{N}$ is a constant specified by the user, and $K_I^p(x, x')$ is a standard single-instance kernel. This is a variant of the set kernel. X and X' are bags of instances. Gärtner et al. (2002) show that any MI problem can be separated using an MI kernel with sufficiently large p.

In addition, another simple MI kernel approach called the "statistic kernel" is also proposed in (Gärtner, Flach, Kowalczyk & Smola, 2002). The idea is to summarize a set (bag) of instances in a single-instance version by computing statistics on the data. In other words, the idea is to simply use statistics, such as the mean, median, maximum, minimum etc., to represent each set (bag) of instances. The statistic kernel is defined as follows:

$$K_{stat}(X, X') := K(s(X), s(X')),$$

where, $s(X)$ is the result of a statistic computed on train bag X, which is represented as a single instance and $K(.,.)$ is a standard single-instance kernel function. For example, if we choose the maximum statistic, $s(X)$ is a vector containing the maximum value of each attribute within bag X.

In this thesis, we upgrade two commonly used kernels implemented in WEKA (the polynomial kernel and the RBF kernel) into MI kernels. Using this transformation, the SVM implementation in WEKA, based on the *SMO* algorithm (Platt, 1998) can be easily applied to MI datasets.

A bag-level polynomial kernel for MI problems can be defined as:

$$K_{\mathrm{MI}}(X, X') = \sum_{x \in X, x' \in X'} (x \cdot x')^d$$

where X and X' represent a pair of bags. x and x' are the individual instances within the bags X and X' respectively. d is the exponent of the polynomial kernel.

Similarly, a bag-level RBF kernel for MI problem can be defined as:

$$K_{\mathrm{MI}}(X, X') = \sum_{x \in X, x' \in X'} \exp(-\gamma \parallel x - x' \parallel^2).$$

Note that in these two cases, we do not need to consider the constant exponent p from Equation 2.7 separately. This can be done equivalently by properly adjusting the exponential parameter d in the polynomial kernel and the parameter γ in the RBF kernel. It is not necessary to involve the new exponent p once again.

Furthermore, we also implement a statistic kernel called "minimax kernel" with the mapping $s(X)$ defined as follows (Gärtner, Flach, Kowalczyk & Smola, 2002):

$$s(X) = (\min x_1, ..., \min x_m, \max x_1, ..., \max x_m),$$

where m is the number of attributes in the original feature space and min and max return the minimum and maximum value in a bag respectively. It is clear that the transformed feature space contains $2m$ attributes.

Apart from its promising performance in experiments (Gärtner, Flach, Kowalczyk & Smola, 2002), the great advantage of this statistic kernel is its efficiency.

2.2.2 MISVM

The MI kernel-based SVM approach described before actually ignores the standard MI assumption because all instances in a bag contribute equally to the bag's label. However, Stuart Andrews et al. (2002) suggested another SVM based approach, which follows the standard MI assumption. The main idea of this approach is to transform the MI data setting into a single-instance data setting by properly assigning the unobserved class label to each individual instance in the positive bags (all instances in negative bags are assumed to be negative). The standard single-instance SVM learning scheme is applied to assign these labels. Andrews et al. call this method "maximum pattern margin formulation of MI learning". The aim is to find the maximum margin MI-separating hyperplane, in which all instances in every negative bag are located on one side of the hyperplane and at least one instance in every positive bags is located on the other side of the hyperplane.

We implemented this approach for the experiments in this thesis. It is called "*MISVM*" in the following. For the standard SVM, we use the *SMO* algorithm in WEKA. The *MISVM* algorithm is given in Figure 2.2.

The key problem in the *MISVM* approach is how to determine and assign the proper class label to the individual instances within each positive bag. The *MISVM* algorithm first initializes all instances in the positive bags with the positive class labels and then iteratively adjusts these labels until they converge.

First, we convert the MI dataset into the single-instance setting by assigning a binary class label (either 1 or 0) to each individual instance according to its corresponding bag label. If an instance belongs to a positive bag, we initialize its class label as "1". Otherwise, if it belongs to a negative bag, the initialized label is "0".

Of course, under the MI assumption, single-instance label settings within the positive bags assigned in this way are not precise enough. The next step is to use the standard SVM algorithm to refine these label settings. This can be achieved by first using the standard SVM algorithm to build an SVM model based on the converted data with initialized labels and then using the obtained SVM model to test each instance in the positive bags. If the original assigned class label does not conform to the result of testing, the original label is changed. Note it is not necessary to test the instances in the negative bags as the MI assumption implies that they all must be negative, which is the same as what they all have been initialized to. In order to ensure that at least one instance within each positive bag is assigned the positive label "1", we should avoid the situation that all have been assigned the negative label "0". Hence, for each positive bag, the sum of the instances' class value should be greater than 0. Otherwise, the algorithm simply chooses the instance that is most likely to be positive and resets its class label to be "1". This can

```
Build Classifier:
For each bag $B$
    If $B$ is a positive bag
        Initialize class label for each instance $x_i$ within $B$ as $y_i = 1$;
    Else
        Initialize the label of each instance $x_i$ within $B$ as $y_i = 0$;
Do
    Build standard single-instance SVM model based on the labeled data;
    For each positive bag $B^+$
        For each single-instance $x_i$ within $B^+$
            Compute SVM output $f(x_i) = \sum_j \alpha_j K(x_j, x_i) + b$;
            If $(f(x_i) \leq 0)$    $y_i = 0$;
            Else $y_i = 1$;
        If $(\sum_i y_i == 0)$ //no positive classification
            Find instance $x_{i^*}$ within bag $B^+$ where $i^* = \arg\max_i f(x_i)$;
            Set $y_i^* = 1$;
While (single-instance labels have been changed)


Classify:
Initialize $distribution$;
For each single-instance $x_i$ within the unknown bag
    Compute $f(x_i) = \sum_j \alpha_j K(x_j, x_i) + b$;
    If $(f(x_i) \leq 0)$    $y_i = 0$;
    Else $y_i = 1$;
If $(\sum_i y_i == 0)$
    $distribution[0] = 1.0$; // predicted as a negative bag
Else
    $distribution[0] = 0$; // predicted as a positive bag
$distribution[1] = 1 - distribution[0]$;
return $distribution$;
```

Figure 2.2: *MISVM* algorithm, adapted from (Andrews, Tsochantaridis & Hofmann, 2002)

be determined by comparing the output values of the currently obtained SVM function (Equation 2.6) for all instances in the bags and picking the one with maximum output value. The algorithm then rebuilds the SVM model on the data with modified class labels. This process of assigning labels is simply repeated until no more label changes are required for any of the instances in the positive bags.

At testing time, a bag-level classification result can be obtained based on instance-level classification results. To obtain instance-level classifications, the resulting SVM model is used to classify each instance within the bag with unknown label and an estimate of class label is obtained for each instance. Then to classify the bag, we simply follow the standard MI assumption: if all instances have a negative class label assigned to them, the unknown bag is classified as a negative bag; otherwise, it is classified as a positive bag.

2.3 Distance-based Approaches

In traditional lazy learning, such as the *k-nearest-neighbor* (KNN) approach, the nearest-neighbors are chosen by measuring the Euclidean distance between points (instances). The unknown class label of an instance can then be predicted as the most common class label of the instances' nearest neighbors. The gist of this type of learning is to make a decision based on similarity. However, for MI problems, the simple Euclidean distance measure scheme is no longer applicable as each point's label is unobserved. So the similarity function needs to be upgraded.

This section discusses three different distance-based approaches that have been proposed to solve MI problems. First, we look at a method called *Citation KNN* (Wang & Zucker, 2000) that introduces a way to measure the distance between bags using the *Hausdorff Distance* (Edgar, 1990) and the final decision making is based on both "references" and "citers". Secondly, we briefly review another method called *Nearest Distribution* (Xu, 2001). Unlike other distance-based approach, this method does not measure the similarity of the observed data directly. Instead, it uses *Kullback-Leibler* to measure the similarity of probability distributions that are derived from the observed data. Thirdly, we discuss a method called *Optimal Ball* (Auer & Ortner, 2004). The *Optimal Ball* method is not a lazy learning approach, but it is similarity based. It aims to find a "ball" in the feature space so that all negative bags are outside the ball and all positive bags are inside or at least intersection with the ball. It introduces a simple way to measure the distance between one single point and a bag.

2.3.1 Citation KNN

Citation KNN uses the *Hausdorff Distance* (Edgar, 1990) to measure the distance between two sets. However, in the original *Hausdorff Distance* method, an outlier data point that is far away from the other points tends to dominate the distance result and thus it is modified. If we have two sets of data $A = \{a_1, ..., a_m\}$ and $B = \{b_1, ..., b_n\}$, the *Hausdorff Distance* is defined as follows:

$$H(A, B) = \max\{h(A, B), h(B, A)\}$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} \parallel a - b \parallel .$$

Suppose the resulting distance between two sets is d using this distance measure. Then d is the minimum value so that each point in one set (A set or B set) can find at least one point in the other set (B set or A set) within Euclidean distance d. However, the two max notations in the definition above tend to determine the distance between the two sets based on outlier data points. In order to avoid this, a modified *Hausdorff Distance* has been proposed (Wang & Zucker, 2000). Instead of choosing the maximum distance when computing $h(A, B)$, the modified version ranks the distances first and chooses the kth distance value. Formally, it modifies the *Hausdorff Distance* definition as follows (Wang & Zucker, 2000):

$$h_k(A, B) = kth_{a \in A} \min_{b \in B} \parallel a - b \parallel .$$

Here, a total of m distance values need to be ranked as A contains m points. If the distances are ranked in an increasing order, $k = m$ chooses the maximum one whereas $k = 1$ chooses the minimum one. Actually, the *Minimal Hausdorff Distance* ($k = 1$) picks the nearest pair of points, with each point belonging to a different set, and regards the distance between them as the distance between the two sets. The experimental results (Wang & Zucker, 2000) shows that in general the *Minimal Hausdorff Distance* performs better than the *Maximum Hausdorff Distance*.

Having defined a distance function, it is possible to apply the standard *KNN* method to MI problems now. We may simply predict the class label of an unknown bag as the most common class label of its nearest bags. However, Wang and Zucker (2000) found that the results were not satisfactory. The conjecture is that this is due to the specific character of MI problems where the MI assumption is true, and where the instance labels within each positive bag are unobserved. Suppose in the feature space there are some positive bags that contain a large number of negative instances. Then it is not surprising that some actual negative bags may regard some of these positive bags as their nearest-neighbors, which may finally result in a false positive result.

To reduce this problem, a method called *Citation KNN* has been introduced by Wang and Zucker (2000). Based on the standard *KNN* strategy, this new method is not only interested in a certain bag's "references" (i.e. nearest neighbors), but also its "citers". Here, the term "citers" refers to the other bags that regard this bag as one of their nearest neighbors. The key point of this solution is that it can provide some valuable information as a useful complement. An analogy is that sometimes one cannot make a correct decision by oneself, especially when one only has limited information about a problem and it may be better to consider other people's suggestions as each person may know different aspects of the problem. For MI problems, finding one bag's true nearest neighbors based on the bag itself may not be enough, as similarity may not be judged correctly. Combining the similarity judgments based on the other bags (i.e. the "citers") one can often obtain a more accurate result. This is why the standard *KNN* approach may not be capable to solve MI problems and it may be possible to achieve higher accuracy using the *Citation KNN* approach.

In practice, the *Minimal Hausdorff Distance* measure is suggested to be used in finding the "references" for a particular bag. Finding the "citers" is a little more complicated because they cannot be obtained directly. Let us denote all training bags as $B = \{b_1, ..., b_n\}$. It is easy to get a ranking of similarity scores S_b for a particular bag b according to a given distance measure. Assume we can retrieve the rank of any other bag $\{b_i | b_i \in B, b_i \neq b\}$ in S_b by function $Rank(b_i, b)$. Wang and Zucker (2000) defines the "C-nearest citers" for a certain bag b as follows:

$$Citers(b, C) = \{b_i | Rank(b_i, b) \leq C, b_i \in B\}.$$

At classification time, both the references' labels and the citers' labels for the unknown bag are considered. Suppose R_p denotes the number of "R-nearest references" being positive, C_p denotes the number of "C-nearest citers" being positive, R_n denotes the number of "R-nearest references" being negative and C_n denotes the number of "C-nearest citers" being negative. Then an unknown bag is judged to be positive if $R_p + C_p > R_n + C_n$. Otherwise, it is judged to be negative. Note that if there is a tie, it is better to classify a bag as a negative bag (Wang & Zucker, 2000). The reason is that under the MI assumption false positive errors are more likely to occur than false negative errors.

2.3.2 The Nearest Distribution Method

The basic *Nearest Distribution* method (Xu, 2001) relies on some special assumptions. It assumes that each bag contains enough instances and all dimensions of the data are equally relevant to the classification. Under these assumptions, a distribution can be derived for each dimension of each bag and the obtained distributions can be used directly for classification instead of the original data. The *Nearest Distribution* method does not employ the MI assumption. It assumes that all instances in a positive bag are equally relevant to the bag's classification.

The first step of the *Nearest Distribution* method is to derive the distribution for each bag from the training data. In principle, arbitrary distribution can be used in this approach. However, simply deriving one Gaussian model for each dimension of each bag is considered due to the often limited number of instances in the bags. Before deriving the distributions, all attribute values are scaled to lie between 0 and 1. For the Gaussian model, we just need to compute the mean and the standard deviation for each attribute. We will use these Gaussian distributions to represent the original data. In other words, we can discard the original data and just store the mean, variance and bag label for future use.

The second step of the *Nearest Distribution* method is to find the nearest neighbors for a test bag. The test bag is also required to be represented by Gaussian distributions for each dimension. This can be done in the same way as the training data has been processed. After that, the obtained testing and training distributions are comparable. A mechanism called the *Kullback-Leibler* distance can be used to compare the distributions.

Also known as the relative entropy, the *Kullback-Leibler* (KL) distance is a measure of difference between two random variables. For the continuous case, it can be defined as follows (Cover & Thomas, 1991):

$$KL(f,g) = \int_{-\infty}^{\infty} f(x) \cdot \log \frac{f(x)}{g(x)} dx \tag{2.8}$$

where $f(x)$ and $g(x)$ are the probability densities of the variable that we are comparing.

Based on above definition on two variables, the multivariate continuous case in MI problems can be defined as follows (Xu, 2001):

$$KL(f,g) = \int\int ... \int_{-\infty}^{\infty} [f \cdot \log \frac{f}{g}] dx_1 dx_2 ... dx_n \tag{2.9}$$

where n is the number of the dimensions, f denotes $f(x_1, ..., x_n)$ and g denotes $g(x_1, ..., x_n)$. As stated above, all dimensions are assumed to be independent (i.e. with one univariate Gaussian per dimension). Therefore we can write $f(x_1, ..., x_n) = \prod f(x_i)$ and $g(x_1, ..., x_n) = \prod g(x_i)$.

The *Kullback-Leibler* can be regarded as a kind of distance measure because the KL value is ensured to be positive and $KL(f,g) = 0$ only if $f(x) = g(x)$. However, it is different from a standard distance measure in that $KL(f,g) \neq KL(g,f)$. The asymmetric character makes it sensitive to the assignment of $f(x)$ and $g(x)$ to the two distributions. Xin (2001) used $f(x)$ to represent the test bag and $g(x)$ to represent the training bags.

Based on the definition of the KL distance in Equation 2.9 and the definition of the Gaussian density function, the KL distance can be expressed in a more specific way after some mathematical transformations (Xu, 2001):

$$KL = \sum_j (\log(\sigma_{j2}/\sigma_{j1}) + (\sigma_{j1}^2/2\sigma_{j2}^2) + (\mu_{j1} - \mu_{j2})^2/2\sigma_{j2}^2 - \frac{1}{2}) \tag{2.10}$$

where j denotes the jth dimension, σ_{j1} and μ_{j1} are the two parameters of the Gaussian distribution derived from the test bag and σ_{j2} and μ_{j2} are derived from the training bag. *Nearest Distribution* finds the nearest neighbors for the test bag based on the KL distance. The rest of the classification process is just the same as in the standard *KNN* algorithm.

The basic *Nearest Distribution* method can be further improved by applying attribute selection and cleansing the noisy data. Attribute selection is done by providing each dimension with a weight. The weight vector for each bag is determined by maximizing the distance between bags with different classes and minimizing it otherwise (Xu, 2001). So Equation 2.10 is modified to become:

$$KL = \sum_j (\log(\sigma_{j2}/\sigma_{j1}) + (\sigma_{j1}^2/2\sigma_{j2}^2) + W_{j2}(\mu_{j1} - \mu_{j2})^2/2\sigma_{j2}^2 - \frac{1}{2}), \tag{2.11}$$

where W_{j2} is the weight for attribute j for the relevant training bag. Cleansing noisy data is done by discarding noisy instances. The noisy data in a bag is defined as the instances within a bag close to the instances of other bags with a different class label (Xu, 2001).

2.3.3 Optimal Ball

The *Optimal Ball* method for MI data has been introduced in (Auer & Ortner, 2004) as a weak learner for the boosting (Freund & Schapire, 1996) approach. The basic idea of the method is to find an optimal ball in the feature space, that all negative bags are outside this ball, in other words, the ball can separate the positive and negative concept. The center of the optimal ball is an instance point from one of the positive bags and the radius of the ball is maximized based on the accuracy evaluation. For classification, if each single point within a new bag is outside the optimal ball, this new bag is classified as negative. Otherwise, it is classified as a positive bag. The algorithm for our MI *Optimal Ball* implementation is given in Figure 2.3.

The first step is to calculate the distance from each instance in each positive bag to all other bags. Assume B denotes a single bag in the training data and x denotes a single instance. The distance between an instance and a bag is defined as follows (Auer & Ortner, 2004):

$$d(x, B) = min_{x' \in B} d(x, x'). \tag{2.12}$$

Here, $d(x, x')$ is a distance measure between two instances. The distance between an instance and a bag is defined as the distance from this instance to its closest instance in the bag. In this thesis, we simply use the Euclidean distance to measure the distance between two instances. However, other distance measures can also be applied. In this way, each instance in the positive bags will have a corresponding distance list that contains the distance values from that point to all bags. In our imple-

```
Build Classifier:
//calculate the distance from each point in each positive bag to all other bags
For each positive bag $B_i^+$
    For each instance $x_{ij}$ in positive bag $B_i^+$
      For each bag $B_k$
         Compute the distance from data point $x_{ij}$ to $B_k$ as follows:
            $d(x_{ij}, B_k) = \min_{x' \in B_k} d(x_{ij}, x')$;
            Store $d(x_{ij}, B_k)$ in $D[i][j][k]$;

//try each instance in each positive bag as a ball center
//and find the center with maximum radius that can obtain the highest accuracy
Let $highestAccuracy = 0, Radius = 0$;
For each positive bag $B_i^+$
    For each instance $x_{ij}$ in bag $B_i^+$
      Make a copy of $D[i][j]$ as $D'[i][j]$ and sort it in an increasing order;
      For each element $k'$ in $D'[i][j]$
         $r = (D'[i][j][k'+1] - D'[i][j][k'])/2 + D'[i][j][k']$;

         //evaluate the accuracy:
         Let $weightCount = 0$;
         For each training bag $B_k$
            If $((D[i][j][k] \leq r)$ AND $B_k$ is positive )
              OR $((D[i][j][k] > r)$ AND $B_k$ is negative )
               $weightCount \; += W_{B_k}$ where $W_{B_k}$ is the weight of bag $B_k$;

            //update the ball center with maximum radius and highest accuracy
            If $(highestAccuracy < weightCount)$
              OR $(highestAccuracy == weightCount$ AND $r > Radius)$
               $BallCenter = x_{ij}$;  $highestAccuracy = weightCount$;  $Radius = r$;


Classify:
Initialize $distribution$;
For each instance $x_i$ in the test bag
    Calculate the distance $d$ between $BallCenter$ and Point $x_i$;
    If $(d \leq Radius)$
       $distribution[1] = 1.0$ //predict as positive;
       break;

$distribution[0] = 1 - distribution[1]$;
return $distribution$;
```

Figure 2.3: *OptimalBall* algorithm, adapted from (Auer & Ortner, 2004)

mentation, the distance from an instance to the bag containing that instance is set to zero.

The second step is to construct the candidate balls. We try all possible balls and find the best one according to the evaluation results on the training data. Each point in the positive bags is tried as a temporary ball center in turn. The possible radii for each ball center are generated based on the distances from this ball center to all other bags (which have been computed in the first step). We simply sort these distance values in an increasing order. The possible radii are all the medium values between two neighbors in the list. For example, suppose there are n bags in total and $\{d_1, ..., d_n\}$ is an ordered distance list corresponding to an instance x. We can construct $n-1$ balls in total with a ball center fixed at point x but with different radii. The list of possible radii can be written as $\{(d_1 + \frac{d_2 - d_1}{2}),\ \dots\ , (d_{n-1} + \frac{d_n - d_{n-1}}{2})\}$, with a total of $n-1$ medium values.

The third step is to evaluate all the candidate balls and select the best one, that is the one reaching the highest accuracy with the maximized radius. Suppose w denotes the weight distribution over all bags. The boosting algorithm assigns weights to the bags, and the weighted accuracy is computed. Formally, the radius of the optimal ball for a particular center x is defined as follows:

$$r_o = \max\{r | \max E(h(x, r), w)\}. \tag{2.13}$$

Here $h(x, r)$ represents a candidate ball with x and r being the center and the radius respectively, and $E(h(x, r), w)$ is the accuracy evaluation function. To classify a training bag using a specific candidate ball $h(x, r)$ is easy, we just need to check the distance from the ball center to the bag, which has already been calculated and stored at the very beginning. If the distance is greater than the ball radius r, the bag is classified as negative. Otherwise, it is classified as positive. The overall evaluation on the whole training data is done by computing the sum of the weights of all correctly classified bags. The optimal ball is computed for each center and the final selection is based on the observed error.

In this algorithm, if the total number of bags is n, all instances in the positive bags will be tried as ball centers with $n-1$ possible radii. For each candidate ball, we evaluate the performance on the training data and record the one with the best performance. If there is more than one candidate ball reaching the same highest score, the one with the maximum radius will be the final selection.

2.4 Applying Single-instance Learners to Multi-instance Problems

In order to solve MI problems, many special purpose algorithms have been developed. On the other hand, as there are many existing single-instance learning algorithms, and making use of these algorithms on MI data would be of great value if applying the single-instance learners were possible.

This section describes three methods that aim to apply single-instance learning algorithms to MI problems. The first one is called the *Wrapper* method (Frank & Xu, 2003). It modifies the MI data by assigning the bag label to each instance of the corresponding bag and then builds a propositional classifier based on the modified data. The second method is called *SimpleMI*, and is simple and very fast. It transforms the original MI data into a format where one instance represents one bag. The transformation is based on some summary statistics of the data in the bag. As a result, a single-instance learner can be directly applied to the transformed data. The last method we are going to discuss is *MIBoost* (Xu, 2003), which upgrades the standard AdaBoost method (Friedman, Hastie & Tibshirani, 2000) so that it can process MI data and combine the "weak" hypotheses generated by a single-instance learning algorithm into a "strong" MI classifier.

2.4.1 Wrapper

The *Wrapper* method for MI problems was proposed in (Frank & Xu, 2003). The method is quite straightforward and the experimental results presented by Frank and Xu show that it is competitive with other MI algorithms on the Musk benchmark datasets. According to the paper, there are two key points to ensure the good performance of the *Wrapper* method.

The first key point is that it discards the standard MI assumption and uses the collective assumption which we have encountered before (Sections 2.1.3, 2.1.4 and 2.2.1). In other words, the *Wrapper* assumes that all instances in a bag contribute equally and independently to the bag's label. To satisfy the requirements of propositional (i.e. single-instance) algorithms, the data is represented in an attribute-value format, and the

Wrapper simply assigns the bag label to each instance in a bag.

The second key point is a suitable weighting scheme for the instances because it is inevitable that some bags contain many more instances than others. In order to equally treat all the bags, the *Wrapper* initializes the weight of each instance according to the size of its corresponding bag. If n_i denotes the number of instances within the ith bag, then the weight of each instance of this bag is set to $\frac{1}{n_i}$. However, this modification is not perfect as some learning algorithms are sensitive to the *absolute* value of the instance weights (Frank & Xu, 2003). The above weight setting scheme may create a large number of instances with a very small weight. Frank and Xu finally propose to solve this problem by multiplying the weight of each instance by a constant factor, letting the total weight of all instances be the same as the number of instances. Therefore, the weight for the jth instance in the ith bag is initialized as follows:

$$w_{ij} = \frac{m}{N} \times \frac{1}{n_i}, \tag{2.14}$$

where m denotes the total number of instances in the data, N denotes the total number of bags in data and n_i denotes the number of instances in the ith bag. Frank and Xu show that applying the above two steps is necessary to achieve good performance on the Musk data.

At prediction time, the obtained model is used to process each instance within an unknown bag and returns a set of class probability estimates for each instance. The question is how to combine these estimates to form a prediction for a bag. There are basically two options based on different assumptions.

The first option is to use the standard MI assumption to guide the final decision. Under the MI assumption, one just needs to check what is the maximum probability for any single instance being positive. If the highest probability is greater than 0.5, that means at least one instance within a bag is predicated to be positive. Therefore, the bag is classified as positive. Otherwise, if the highest probability of being positive is less than 0.5, it means all instances within a bag are predicated to be negative. Therefore, the bag is classified as negative. It seems natural to make the prediction in this way for MI problems. However, the experimental results on the Musk data are not satisfactory (Frank & Xu, 2003).

On the other hand, as suggested in (Frank & Xu, 2003), sticking to the collective assumption would mean making each instance-level probability estimate contribute equally to the bag-level prediction. Thus, under the collective assumption, the final decision should be made based on the average of the class probabilities of all individual instances. The average can be either the arithmetic or the geometric average. The experimental results show the accuracy is higher using the collective assumption on Musk data (Frank & Xu, 2003). This makes sense because the collective assumption is used at training time and should thus be used at prediction time.

The *Wrapper* provides an easy solution to use the various single-instance learners to solve MI problems. The only requirement is that the single-instance learner should be able to deal with instance weights and provide the class probability estimates.

2.4.2 Simple MI

SimpleMI is another method that can directly apply single-instance learning algorithms to MI problems. Compared with other similar methods, *SimpleMI* is a very simple one. Apart from its simplicity, it has another significant advantage in that it is very fast.

The idea is to summarize each bag of data and construct one instance to represent the whole bag. Of course, the instance label is inherited from the corresponding bag. Through the summarizing process, the MI dataset is converted into a single-instance dataset and any kind of single-instance learner can be applied to it without any constraint. In this way, a model can be efficiently obtained based on the summarized data.

At prediction time, we first summarize the unknown bag in the same way as it has been done for the training bags. Then we use the obtained single-instance model to predict the class label directly. Obviously, this "one-instance-representing-one-bag" style makes the prediction process very straightforward by avoiding the combination process required in other methods.

For this thesis, three different summarization methods have been implemented in *SimpleMI*. The first one is named "arithmetic mean". For each bag, it computes the arithmetic mean of each attribute and then constructs a new instance where the value of each attribute is set to the mean. The second option is named "geometric mean" and computes the geometric mean of each attribute for every bag. Formally, if n is the total number of dimensions and $\bar{x}_1$ refers to the mean value of the first dimension, a new instance used to summarize a bag is $(\bar{x}_1, ..., \bar{x}_n)$.

The third option is named "minimax" and records both the minimum and maximum value of each dimension for every bag. Actually, we have already encountered this kind of summarization when discussing the MI kernel in Section 2.2.1. It can be expressed as $(\min x_1, ..., \min x_n, \max x_1, ..., \max x_n)$, where n is the number of dimensions in the original data. It is clear that the transformed data contains $2n$ dimensions.

2.4.3 MI Boosting

The boosting approach is well known for excellent performance in the single-instance setting. Its most commonly used version called *AdaBoost* was first introduced in (Freund & Schapire, 1996). Boosting sequentially generates a set of weak classifiers by reweighting the training instances and as a result a much stronger classifier can be obtained by combining these weak classifiers. It encourages new weak classifiers to become experts for instances handled incorrectly by earlier ones (Witten & Frank, 2005). If we have a weak learner that can deal with MI problems and handle bag weights, the *AdaBoost* algorithm can be applied directly to MI problems. For example, using the MI learner *Optimal Ball* described in Section 2.3.3 as a weak learner, *AdaBoost* can easily be applied without any changes.

However, due to the limited number of MI learners, it is worthwhile to find a way to wrap the boosting algorithm around a single-instance weak learner. Motivated by this point, Xu and Frank (2004) upgraded the *AdaBoost* algorithm into an MI learner called *MIBoost* based on the collective assumption. It has been shown that boosting can be viewed and understood under some statistical principles, namely additive modeling and maximum likelihood (Friedman, Hastie & Tibshirani, 2000). The *MIBoost* algorithm follows the same statistical principles. Like *AdaBoost*, *MIBoost* strives to find an additive model which can be expressed iteratively as

$$F(B) = F(B) + cf(B). \tag{2.15}$$

```
1. Initialize weights of each bag to $W_i = 1/N, i = 1, 2, ..., N$.

2. Repeat for $m = 1, 2, ..., M$:
    (a) Set $W_{ij} \leftarrow W_i/n_i$, assign the bag's class label to each
        of its instances, and build an instance-level model $h_m(x_{ij}) \in \{-1, 1\}$.
    (b) Within the $i$th bag (with $n_i$ instances), compute the error rate $e_i \in [0, 1]$
        by counting the number of misclassified instances within that bag,
        i.e. $e_i = \sum_j 1_{(h_m(x_{ij}) \neq y_i)}/n_i$.
    (c) If $e_i < 0.5$ for all $i$'s, STOP iterations, Go to step 3.
    (d) Compute $c_m = \arg\min \sum_i W_i \exp[(2e_i - 1)c_m]$ using numeric optimization.
    (e) If $(c_m \leq 0)$, go to step 3.
    (f) Set $W_i \leftarrow W_i \exp[(2e_i - 1)c_m]$ and renormalize so that $\sum_i W_i = 1$.

3. return $sign[\sum_i \sum_m c_m h_m(x_{test})]$.
```

Figure 2.4: *MIBoost* algorithm, adapted from (Xu & Frank, 2004)

Here, B denotes an arbitrary bag, $f()$ denotes a weak classifier and c is a constant that needs to be learned in each boosting iteration. The details of the derivation of the *MIBoost* algorithm are described in (Xu & Frank, 2004). In the following, we describe the basic steps of the *MIBoost* algorithm.

Figure 2.4 shows the pseudo code for *MIBoost* algorithm. Here, N is the total number of bags, n_i denotes the number of instances in a bag and the subscript i denotes the ith bag. h_m denotes a single-instance weak classifier built in the mth iteration. The value of $h_m()$ is either -1 or 1. Note that, for convenience, Xu and Frank assume the class label of a bag is either 1 or -1 rather than 1 or 0 here. x_{ij} denotes the jth instance in the ith bag and y_i denotes the class label of the ith bag.

MIBoost begins by equally initializing the weights of each bag W_i to $\frac{1}{N}$ and then it iteratively generates a set of single-instance weak classifiers based on the collective assumption. Using the collective assumption, *MIBoost* assigns equal weights ($\frac{W_i}{n_i}$) and the same bag label to all instances within a bag. Therefore, a single-instance weak classifier h_m can be built as shown in Step 2(a) of Figure 2.4. The next step is to evaluate the weak learner h_m on the training data. The evaluation is based on the error rate of each bag $e_i \in [0, 1]$, which can be obtained by counting the number of misclassified instances within that bag (i.e. $e_i = \sum_j 1_{(h_m(x_{ij}) \neq y_i)}/n_i$). According to the collective assumption, all instances in a bag contribute equally and independently to the bag label. The bag-level decision can be determined by the majority prediction for all instances within a bag. If

e_i is less than 0.5, it means the bag-level prediction of the ith bag is correct. Therefore, having all e_i less than 0.5 for all bags is equivalent to correctly predicting all the bags. If this is the case, the iterations can be stopped as indicated in Step 2(c) of Figure 2.4. On the other hand, if this is not the case, the next key step is how to update the bag-level weights W_i according to the error rate e_i.

As we know, in each boosting iteration, *MIBoost* adds a classifier $f()$ with the best value for c to the model. The aim is to minimize the exponential loss $E_B E_{Y|B}[\exp(-yF(B))]$. More precisely, in the mth iteration, $f_m(B)$ is obtained from the weighted version of the training data, and then c_m is derived by minimizing the exponential loss function (Xu & Frank, 2004):

$$E_B E_{Y|B}[\exp(-yF(B) + c_m(-yf_m(B)))] = \sum_i W_i \exp[c_m(-\frac{y\sum_j h(x_{ij})}{n_i})]$$

$$= \textstyle\sum_i W_i \exp[(2e_i - 1)c_m].$$

In the implementation, c_m is found using a Quasi-Newton method (Xu & Frank, 2004). This is done in Step 2(d). Note that in Step 2(e), c is constrained to be positive just in order to act in accordance with *AdaBoost* (Xu & Frank, 2004).

After the best value for c has been obtained, the bag-level weights can be updated. Each W_i is multiplied by $\exp[(2e_i - 1)c_m]$ and all the bags' weights are normalized so that the total weight remains 1.

It has been shown (Friedman, Hastie & Tibshirani, 2000) that in *AdaBoost*, the additive model obtained by minimizing the exponential loss function estimates the log-odds $\frac{1}{2}\log\frac{Pr(Y=1|X)}{Pr(Y=-1|X)}$. This is the same in *MIBoost*. Hence a prediction is made by treating the obtained model $F(B)$ as a bag-level log-odds function. This is implemented using the simple summation in Step 3.

2.5 Two-level Distribution Approach

The two-level distribution approach (*TLD*) was originally proposed by (Xu, 2003). The idea is to extract distribution parameters for each bag and to model these parameters using bag-level distributions, so that the class probabilities can be obtained using Bayes rule.

2.5.1 TLD

The *TLD* method contains two steps. In the first step, it derives the distribution properties from all individual instances within each bag. The obtained distributions are the so-called "instance-level distributions". In the second step, it aims to derive hyper-distributions from these instance-level distributions so that the positive and negative classes can be discriminated. The distributions obtained from the second step are called the "bag-level distributions".

Now we briefly discuss how to derive these two-level distributions. The *TLD* method discards the MI assumptions and uses the collective assumption instead. It assumes all individual instances within a bag contribute equally and independently to the bag label. Under this assumption, the instance-level distributions can be obtained using arbitrary distribution models. However, for computational tractability, Xu (2003) uses a Gaussian model for each dimension of each bag. Then the question is how to derive bag-level distributions based on the instance-level distributions. According to (Xu, 2003), the problem can be considered from a Bayesian perspective (O'Hagan, 1994). For each class, we can regard the Gaussian parameters of the instance-level distributions as random and governed by a hyper-distribution. Therefore, the task is converted into estimating the parameters of the hyper-distributions (i.e. bag-level distributions) for the different classes.

We use the notation introduced in (Xu, 2003). Let us denote the jth bag with n_j instances as $b_j = \{x_{j1}, ..., x_{jk}, ..., x_{jn_j}\}$. θ denotes the parameters of the instance-level distributions. Y denotes the class label which can be either 0 or 1 in the two-class case. δ^y denotes the bag-level distribution parameters for each class ($y = 1$ for the positive class and $y = 0$ for the negative class). Therefore, $Pr(b_j|Y)$ can be written as $Pr(b_j|\delta^y)$. The likelihood function L can be written as follows:

$$L = \prod_j Pr(b_j|\delta^y) = \prod_j \int Pr(b_j|\theta)Pr(\theta|\delta^y)\ \mathrm{d}\theta. \tag{2.16}$$

Here, $Pr(b_j|\theta)$ represents an instance-level distribution. If we assume all individual instances within a bag are independent, we have $Pr(b_j|\theta) = \prod_i^{n_j} P(x_{ji}|\theta)$ where x_{ji} denotes the ith instance in the jth bag. If there are m attributes and e bags for a particular class, and we further assume all the attributes are independent, the likelihood function can be rewritten as:

$$L = \prod_{j=1}^{e}(\prod_{k=1}^{m}\{\int[\prod_{i=1}^{n_j} Pr(x_{jik}|\theta_k)]Pr(\theta_k|\delta_k^y)\ \mathrm{d}\theta_k\}). \tag{2.17}$$

As mentioned before, a Gaussian model is used to estimate instance-level probability $Pr(x_{jik}|\theta_k)$ with mean μ_k and variance σ_k^2. So we have:

$$\begin{aligned}\prod_{i=1}^{n_j} Pr(x_{jik}|\theta_k) &= \prod_{i=1}^{n_j}((2\pi\sigma_k^2)^{-1/2}\exp[-\frac{(x_{jik}-\mu_k)^2}{2\sigma_k^2}]) \\ &= (2\pi\sigma_k^2)^{-n_j/2}\exp[-\frac{S_{jk}^2+n_j(\bar{x}_{jk}-\mu_k)^2}{2\sigma_k^2}] \qquad (2.18)\end{aligned}$$

where $\bar{x}_{jk} = \sum_{i=1}^{n_j} x_{jik}/n_j$ and $S_{jk}^2 = \sum_{i=1}^{n_j}(x_{jik}-\bar{x}_{jk})^2$.

For the bag-level probability $Pr(\theta|\delta_k^y)$ in Equation 2.17, we apply the corresponding *natural conjugate* form of the Gaussian distribution (O'Hagan, 1994). As described in (Xu, 2003), the natural conjugate form with four parameters (a_k, b_k, w_k and m_k) can be written as follows:

$$Pr(\theta_k|\delta_k^y) = g(a_k,b_k,w_k){\sigma_k^2}^{-\frac{b_k+3}{2}}\exp(-\frac{a_k+\frac{(\mu_k-m_k)^2}{w_k}}{2\sigma_k^2}) \tag{2.19}$$

where

$$g(a_k,b_k,w_k) = \frac{a_k^{\frac{b_k}{2}}2^{-\frac{b_k+1}{2}}}{\sqrt{(\pi w_k)}\Gamma(b_k/2)}.$$

In this bag-level model, μ_k follows a normal distribution with mean m_k and variance $w_k\sigma_k^2$. The variance σ_k^2 follows an *Inverse-Gamma* distribution (O'Hagan, 1994). After combining the two levels' models (Equations 2.18 and 2.19), and combing the integral in Equation 2.17, the likelihood function for one class can be written as (Xu, 2003):

$$L = \prod_{j=1}^{e}\prod_{k=1}^{m}\frac{a_k^{b_k/2}(1+n_jw_k)^{(b_k+n_j-1)/2}\Gamma(\frac{b_k+n_j}{2})}{[(1+n_jw_k)(a_k+S_{jk}^2)+n_j(\bar{x}_{jk}-m_k)^2]^{\frac{b_k+n_j}{2}}\pi^{\frac{n_j}{2}}\Gamma(\frac{b_k}{2})}. \tag{2.20}$$

The corresponding log-likelihood function is:

$$LL = \sum_{k=1}^{m} \sum_{j=1}^{e} \log B_{jk}$$

where

$$B_{jk} = \frac{a_k^{b_k/2}(1+n_j w_k)^{(b_k+n_j-1)/2}\Gamma(\frac{b_k+n_j}{2})}{[(1+n_j w_k)(a_k+S_{jk}^2)+n_j(\bar{x}_{jk}-m_k)^2]^{\frac{b_k+n_j}{2}}\pi^{\frac{n_j}{2}}\Gamma(\frac{b_k}{2})}. \tag{2.21}$$

The *TLD* approach learns the four parameters (a_k, b_k, w_k and m_k) by maximizing this log-likelihood function. This is done using the quasi-Newton optimization procedure with BFGS updates implemented in WEKA (Xu, 2003). It can be seen from Equation 2.21 that the log-likelihood function only involves the sample mean $\bar{x}_{jk}$ and sum of squared errors S_{jk} for bag j and attribute k.

At classification time, we simply compute the mean and the sum of squared errors of the new bag b and then compute the log-odds function:

$$\log \frac{Pr(Y=1|b)}{Pr(Y=0|b)} = \log \frac{Pr(b|\delta^1)Pr(Y=1)}{Pr(b|\delta^0)Pr(Y=0)} \tag{2.22}$$

where the two prior probabilities $Pr(Y=1)$ and $Pr(Y=0)$ can be estimated from the training data according to the number of bags for each class. If the resulting log-odds value is greater than zero, the new bag is classified as positive ($Y=1$). Otherwise, it is classified as negative ($Y=0$).

2.5.2 TLD Simple

In practice, running the original *TLD* method described above can take a significant amount of time as four parameters per attribute are required to be estimated through the optimization procedure. To solve this problem, a simplified approach called *TLD Simple* has also been proposed in (Xu, 2003). Two modifications have been made in this simplified version:

- In the instance-level distribution (Equation 2.18), the S_{jk} term is dropped. This modification is based on the well known "central limit theorem", which states that the mean of any set of variates with any distribution having a finite mean and variance tends to the normal distribution (Feller, 1971). Dropping the S_{jk} is equivalent to assuming that the mean $\bar{x}_{jk}$ follows a normal distribution with parameters μ_k and

$\frac{\sigma_k^2}{n_j}$.

- In the bag-level distribution (Equation 2.19), the "Inverse-Gamma" distribution for σ_k^2 is dropped. Instead, σ_k^2 is regarded as fixed and is directly estimated from the training data. Thus the parameters a_k and b_k are no longer needed. σ_k^2 is estimated as follows (Xu, 2003):

  $$\hat{\sigma_k^2} = \sum_j [\frac{1}{n_j} \sum_i (x_{jik} - \bar{x}_{jk})^2] / (e - \sum_j \frac{1}{n_j})$$

  where n_j is the number of instances in the jth bag and e is the number of bags. This is the average of the per-bag sample variances.

After applying these two simplifications, the log-likelihood function for one class becomes (Xu, 2003):

$$LL = \sum_{k=1}^{m} \sum_{j=1}^{e} \log B_{jk}$$

where

$$B_{jk} = (2\pi \frac{w_k n_j + \sigma_k^2}{n_j})^{-1/2} \exp[\frac{-n_j(\bar{x}_{jk} - m_k)^2}{2(w_k n_j + \sigma_k^2)}]. \tag{2.23}$$

Compared with Equation 2.21, the parameters a_k and b_k have disappeared. Moreover, the maximum likelihood solution of m_k has a simple analytical form so that only the parameter w_k needs a numeric optimization procedure (Xu, 2003). The required computing time is thus dramatically decreased.

Chapter 3

Applications and Experiments

This chapter evaluates the MI algorithms described in Chapter 2 on a set of real-world datasets. As standard MI learning is only defined for two-class problems, we only consider two-class problems. All experiments were performed using the experiment environment of the WEKA workbench (Witten & Frank, 2005). The datasets used represent a wide range of application domains. We first introduce these application domains and then briefly describe our experimental methodology. After that, we show and compare the experimental results of the different MI algorithm approaches. The aim is to investigate what kind of algorithm is suitable for what type of MI problem and to determine suitable MI approaches for each specific application domain.

3.1 Multi-instance Application Domains

In this section, we introduce the application domains involved in our experiments. They include drug activity prediction, image retrieval, protein identification, the classic East-West challenge and text categorization. Some of them, such as the East-West challenge, originally come from Inductive Logic Programming (ILP) tasks. Using the Proper toolbox (Reutemann, 2004), the relational data can be transformed into a multi-instance representation by flattening the nested structure of the relational data into a single table with multiple instances per bag. As a result, a total of 15 datasets where used in the experiments. Tables 3.1 and 3.2 list the key properties of these datasets. Note the number of attributes does not include the bag ID and the class attribute.

3.1.1 Drug Activity Prediction

The most popular MI application is the drug activity prediction problem introduced in (Dietterich, Lathrop & Lozano-Perez, 1997), represented by the Musk datasets. The Musk datasets are the standard benchmark datasets in the MI domain. Almost all papers

Table 3.1: Properties of the twelve MI Datasets

Dataset	Number of Bags			Number of Attributes	Number of Instances	Average Bag Size
	Positive	Negative	Total			
Musk1	47	45	92	166	476	5.17
Musk2	39	63	102	166	6598	64.69
Mutagenesis-atoms	125	63	188	10	1618	8.61
Mutagenesis-bonds	125	63	188	16	3995	21.25
Mutagenesis-chains	125	63	188	24	5349	28.45
Suramin	7	4	11	20	2378	216.18
Elephant	100	100	200	230	1391	6.96
Tiger	100	100	200	230	1220	6.10
Fox	100	100	200	230	1320	6.60
Trx	25	168	193	8	26611	137.88
EastWest	10	10	20	24	213	10.65
WestEast	10	10	20	24	213	10.65

Table 3.2: Properties of the three MI Datasets with an explicit test set

Dataset	Training Bags			Testing Bags			Number of Attributes	Number of Instances	Average Bag Size
	Pos.	Neg.	Toltal	Pos.	Neg.	Total			
Component	359	359	718	64	2348	2412	200	36894	11.79
Function	385	385	770	58	4414	4472	200	55536	10.59
Process	620	620	1240	137	10341	10478	200	118417	10.11

that introduce new MI algorithms have used these datasets to perform the evaluation. In the Musk datasets, a single bag represents one molecule. Each molecule can have several different conformations (i.e. shapes), and a single instance within a bag represents one conformation of the corresponding molecule. The molecules in the Musk datasets have been assigned labels as either being of "musk" type or "non-musk" type by human experts. However, it is unknown which kind of conformation of a molecule results in the "musk" label (i.e. the true class labels of the individual instances are unknown). The aim is to predict whether a new molecule is of "musk" or "non-musk" type. The Musk problem has two versions, Musk1 and Musk2. Compared with the Musk1 dataset, molecules contain more conformations in the Musk2 dataset.

Another drug activity prediction problem we consider is the mutagenicity of molecules. The original Mutagenesis dataset (Srinivasan, Muggleton, King & Sternberg, 1994) represents a relational problem and has been widely used in the ILP domain. The prediction of the mutagenicity of a compound molecule is related to the prediction of carcinogenesis. The original relational data can be represented as MI data. As explained in (Reutemann, Pfahringer & Frank, 2004; Reutemann, 2004), the relational data can be flattened into a single table by performing joins on the original tables. This method was used to transform the relational data into an MI dataset. In the resulting MI dataset, each bag represents one compound molecule. Three datasets were generated using the following three transformations proposed in (Reutemann, 2004) :

a) each bag contains all atoms of a compound molecule

b) each bag contains all atom-bond tuples of a compound molecule

c) each bag contains all adjacent pairs of bounds of a compound molecule

The last drug activity prediction problem we consider is the Suramin dataset (Braddock, Hu, Fan, Stratford, Harris & Bicknell, 1994), which also comes from the ILP domain. It uses the atomic structure and bond relationships to represent a compound molecule. The task is to discover whether a compound can be active or inactive as an anti-cancer agent. Like the Mutagenesis problem, this ILP problem was transformed into an MI problem by flattening the relational data into a single table (Reutemann, Pfahringer & Frank, 2004; Reutemann, 2004). Note that the Suramin dataset contains only 11 bags.

3.1.2 Image Retrieval

The MI model has been used in content-based image retrieval (CBIR) (Maron & Ratan, 1998; Maron & Lozano-Perez, 1998; Zhang, Yu, Goldman & Fritts, 2002) and recently several researchers have evaluated MI algorithms on image datasets (Andrews, Tsochantaridis & Hofmann, 2002; Tao, Scott & Vinodchandran, 2004; Ray & Craven, 2005). In CBIR, users present examples of their desired images and the task is to figure out whether a given image is one they are interested in. An image is represented by a set of segments (pixel regions) that are characterized by color, texture and shape descriptors. A bag represents an image and an instance within a bag represents one segment of the image. It is unknown which segments and features of an image are related to the desired content (i.e. the class labels of the individual instances are again unknown). In the experiments presented here, three animal image datasets ("elephant", "fox" and "tiger") provided by (Andrews, Tsochantaridis & Hofmann, 2002) were used. If an image contains the desired animal, it is a positive example. Each dataset contains 100 positive examples (bags) and 100 negative examples (bags). The negative examples are randomly drawn from the other categories. The goal is to distinguish images containing the desired animal from those that do not contain it.

3.1.3 Identifying Trx-fold Proteins

In tasks involving the identification of new proteins in superfamilies with low primary sequence conservation, such as the Thioredoxin-fold (Trx-fold) protein identification task, conventional approaches (e.g. building hidden Markov models on the primary sequence data) become inefficient (Wang, Scott, Zhang, Tao, Fomenko & Gladyshev, 2004). Therefore, Wang et al. proposed the idea of using multi-instance learning to solve the protein identification problem. Later, identification of Trx-fold protein has been regarded as a multi-instance problem by others, such as in (Tao, Scott & Vinodchandran, 2004; Ray & Craven, 2005).

In the Trx-fold protein problem the aim is to identify whether a given protein belongs to the family of Trx proteins. It has been framed as a multi-instance problem in the following way (Tao, Scott & Vinodchandran, 2004). First, all the sequence patterns, called "motifs", in each protein's primary sequence are found. After that, all fixed-length subsequences around the motifs are extracted (30 upstream, 180 downstream, with total length 214). Then each extracted sequence is represented by eight numeric properties. Finally, each protein is represented as a bag and each extracted sequence is represented as an instance within the bag.

3.1.4 East-West Challenge

The well-known East-West Challenge was originally presented in the ILP domain (Michalski & Larson, 1977). The task is to predict whether a train is eastbound or westbound. A train contains a variable number of cars which have different shapes and carry different loads. The problem is represented using a relational description format. As we have encountered several times before, the ILP problem with a relational representation can be transformed into the MI setting by flattening the data into a single table.

As the standard MI assumption is asymmetric and it is not clear whether an eastbound train or a westbound train can be regarded as a positive example in the MI setting, we generated two MI versions of this dataset for our experiments. One is called "East-West", in which the eastbound trains are positive examples and the other is called "West-East", in which the westbound trains are positive examples.

3.1.5 Text Categorization

We use the text categorization domain introduced in (Ray & Craven, 2005). The task is a part of the work described in (Blaschke, Leon, Krallinger & Valencia, 2005). Given a name of a protein and a full-text article from a biomedical journal, the task is to determine whether this protein-article pair can be annotated with a particular Gene Ontology (GO) code. For the MI setting, each article is represented as a bag. An instance of a bag refers to a paragraph of the corresponding article. Each paragraph is represented as a set of word occurrence frequencies and some statistics about the nature of the protein-GO code interaction. The idea is that if there is one instance (paragraph) that is related to the protein-GO code, the bag is positive. Otherwise, if no instance (paragraph) within a bag is related to the protein-GO code, the bag is negative. The GO contains three aspects of gene products, namely *cellular components*, *molecular functions* and *biological processes*. Therefore, three MI datasets (called *component*, *function* and *process*) are generated according to each GO aspect.

In this thesis, we used these three text categorization datasets. Their key properties are listed in Table 3.2. As we can see from the table, the training data and the testing data are separated. For the training data, the number of positive and negative bags is the same. However, for the testing data, the proportion of the positive bags is very small (less than 3%). The training data used to build the model consists of articles published in the *Journal of Biomedical Chemistry* and the testing data used to test the model are articles published in *Nature* (Ray & Craven, 2005).

3.2 Experimental Methodology

For this thesis, extensive experiments on the MI datasets described above with a wide range of MI approaches were performed. Except for the *TLD* method (due to its excessive computing time), all algorithms described in Chapter 2 (including *TLDSimple*) have been evaluated empirically on the MI datasets in Tables 3.1 and 3.2, and these datasets represent a broad range of practical problems.

The repeated hold-out method was used to obtain performance estimates for the learning methods investigated. Except for the three datasets from the text categorization domain, which have pre-defined train/test splits, all the datasets were randomly split into two subsets for training and testing. For each learning scheme, we repeated the random split 100 times. In each split, 90 percent of the data (i.e. using 90% of the bags) were used for training and the remaining 10 percent for testing. The data was stratified at the bag-level to preserve class proportions. Final accuracy estimates were obtained by averaging the 100 runs' results. We also recorded the standard deviation and tested for significant differences using the corrected resampled t-test (Nadeau & Bengio, 2003) at the 5% significance level. This test was specifically designed for the repeated hold-out method.

For the three text categorization datasets, we performed one single run based on the pre-defined train/test split for each scheme investigated. Because the class distribution is different in the train and test sets, the *Area Under Curve* (AUC) measure (Bradley, 1997) was used in addition to accuracy on these three datasets.

All the experiments were performed using the WEKA workbench (Witten & Frank, 2005). All algorithms were applied with their default settings if other parameter settings are not specifically mentioned. For the approaches described in Section 2.4, i.e. applying single-instance learners to MI problems, we used a variety of single-instance learning algorithms implemented in WEKA as the base learners.

3.3 Experimental Results and Analysis

This section presents the experimental results for all the MI algorithms described in Chapter 2. In order to present a clear view, it first shows the results grouped by type of learning algorithm as well as application domain. The results on the three text categorization problems are discussed separately in Section 3.3.6. Then we compare the results obtained from different approaches across domains. Finally, we discuss and look for a suitable approach for each application domain.

For each learning scheme, the classification accuracy with standard deviations is reported (except on the three text categorization datasets). A significant improvement in accuracy for the reference scheme compared to the other schemes is marked with a "$\circ$", and a significant degradation in accuracy is marked with a "$\bullet$".

3.3.1 Results for Probabilistic Approaches

We ran two sets of experiments for the probabilistic approaches. One covers the Diverse Density (DD) methods from Sections 2.1.1, 2.1.2 and 2.1.3, and the other the Logistic Regression (LR) methods from Section 2.1.4.

For the *DD* approach, the following three versions of the *DD* algorithm were evaluated on the 12 datasets from Table 3.1:

- *DD*: standard diverse density algorithm with noisy-or model (Section 2.1.1)
- *EMDD*: Expectation-Maximization version of *DD* with the most-likely-cause model (Section 2.1.2)
- *MDD*: *DD* algorithm under the collective assumption (Section 2.1.3)

The accuracy estimates obtained from 100 runs of the hold-out method are listed in Table 3.3.

Generally, *DD* performs best among the three but the advantage is not significant. From Table 3.3 we can see that 9 out of 12 accuracy results obtained from *DD* are higher than both obtained from the other two algorithms. However, there are no significant differences except on the Mutagenesis-chains dataset. It is surprising that there is no dataset where making the MI assumption yields a significant advantage (*DD* vs *MDD*).

The *EMDD* method appears to be a promising candidate scheme for datasets with large bag size due to its improved computation efficiency compared to *DD*. The performance of *EMDD* on the standard benchmark Musk2 dataset, with an average bag size of 64.69 instances, is somewhat better than that of the *DD* method. More importantly, the accuracy reaches 85.57% for *EMDD* but only 80.39% for *DD*. *EMDD* outperforms *DD* on Musk2 not only in terms of accuracy but also in terms of training time. The average training time of *EMDD* was about 4 times faster than training of *DD* on the Musk2 dataset (1306.67 seconds for *EMDD* and 5060.85 seconds for *DD*). It appears that for datasets with a large bag size, *EMDD* may be a promising alternative to *DD* that can improve efficiency without degrading classification accuracy.

For the logistic regression (LR) approach (Section 2.1.4), the following three versions were evaluated on the 12 datasets from Table 3.1:

- *MILR*: logistic regression using the standard MI assumption
- *MILR-ARITH*: logistic regression using the collective assumption with arithmetic mean

Table 3.3: Accuracy of Variants of the Diverse Density Approach

Dataset	DD	EMDD	MDD
Musk1	84.65±11.83	83.61±13.16	77.64±15.31
Musk2	80.39±13.44	85.57±10.28	73.01±14.20
Mutagenesis-atoms	72.89± 9.34	68.89±12.29	71.74± 7.65
Mutagenesis-bonds	75.96± 7.48	73.43± 9.22	72.48± 7.54
Mutagenesis-chains	79.90± 9.67	71.49±12.22 •	77.05± 9.30
Suramin	65.88±38.74	52.94±41.08	67.06±37.44
Elephant	81.45±10.23	75.25±10.57	78.35± 9.21
Fox	59.40± 9.88	59.65± 9.16	64.80±10.44
Tiger	72.20± 8.80	71.35± 9.45	67.70± 9.36
Trx	89.99± 6.00	87.92± 6.22	87.06± 2.29
EastWest	64.50±32.79	64.00±28.50	56.50±29.86
WestEast	36.00±29.37	38.00±27.63	49.00±33.32

◦, • statistically significant improvement or degradation wrt left-most scheme

Table 3.4: Accuracy of Variants of the Logistic Regression Approach

Dataset	MILR	MILR-ARITH	MILR-GEOM
Musk1	73.51±13.43	85.03±11.26 ◦	76.68±13.88
Musk2	78.51±13.06	81.26±10.85	78.94±12.33
Mutagenesis-atoms	74.00±10.37	71.47± 8.92	69.80±10.15
Mutagenesis-bonds	76.61± 8.40	68.58± 7.08 •	82.96± 7.94
Mutagenesis-chains	78.27± 8.84	72.74± 8.91	78.44± 9.60
Suramin	60.00±39.94	38.82±39.62	67.06±37.44
Elephant	78.90± 9.28	79.85± 9.14	79.75± 8.92
Fox	57.35±10.81	55.20±10.12	52.50± 9.99
Tiger	75.30± 8.98	76.35± 8.64	79.00± 9.24
Trx	85.30± 5.18	87.11± 2.30	87.31± 4.37
EastWest	67.00±36.39	59.00±34.36	55.00±34.45
WestEast	35.50±34.30	57.00±33.35	55.00±34.45

◦, • statistically significant improvement or degradation wrt left-most scheme

- *MILR-GEOM*: logistic regression using the collective assumption with geometric mean

The accuracy estimates for these three versions of the *LR* algorithm are listed in Table 3.4. From these results, it is hard to say which method is the best. In most cases, there are no significant differences among the three methods. The *MILR-ARITH* method obtains one significantly higher accuracy compared to *MILR* on Musk1 and one significantly lower accuracy result on Mutagenesis-bonds. On the two versions of the Musk datasets, the *MILR-ARITH* method performs better than the other two methods. Overall, we can say that there is no clear advantage for the method that attempts to exploit the standard MI assumption (*MILR*).

Comparing *DD* with the three versions of *LR*, we find that *DD* outperforms each *LR* version on 8 out of 12 datasets. Compared with *DD*, the degradation in accuracy on the Musk1 and Trx datasets is significant in the case of *MILR*, and the degradation in accuracy on the Mutagenesis-chains and Mutagenesis-bonds datasets is significant in the case of *MILR-ARITH*. However, the three *LR*-based methods are much faster than the

DD method. In the experiments, the *MILR* was more than 50 times faster than the *DD* method on the musk2 dataset.

Comparing Tables 3.3 and 3.4, we can also see that for the East-West challenge, the EastWest dataset, which regards eastbound trains as positive examples, appears to be a closer fit to the standard MI assumption, compared with the WestEast dataset that regards the westbound trains as positive examples. The accuracy results for the two datasets (EastWest and WestEast) obtained from *DD*, *EMDD* and *MILR*, which follow the standard MI assumption, exhibit big differences. They are 64.50% vs 36%, 64% vs 38% and 67% vs 35.5% respectively. On the other hand, as *MDD*, *MILR-ARITH* and *MILR-GEOM* follow the collective assumption, the results for these three methods on the two datasets are similar or the same. They are 56.5% vs 49%, 59% vs 57% and 55% vs 55%. Thus it appears that east bound trains should be viewed as positive examples.

In summary, *DD* is the most competitive probability-based method for the kinds of MI problems investigated here. However, one obvious drawback of the *DD* method is its expensive running time. For the datasets with large bag size, using *EMDD* can greatly reduce the running time. However, compared with the *DD*-based approaches, the three versions of the logistic regression method are much more efficient.

3.3.2 Results for Support Vector Machine Approaches

In the following, the MI kernel-based SVM method (Section 2.2.1) and the *MISVM* method (Section 2.2.2) are evaluated empirically. Table 3.5 shows the accuracy estimates obtained from 100 runs of the hold-out method on the 12 datasets from Table 3.1. The table compares the following six different schemes:

- *MISMO(RBF)*: MI kernel-based SVM method using MI RBF kernel (the RBF kernel parameter gamma was set to 0.1)
- *MISMO(Poly)*: MI kernel-based SVM method using MI polynomial kernel (the exponent was set to 2)
- *MISMO(minimax)*: MI kernel-based SVM method using minimax kernel combined with polynomial kernel (the exponent was set to 2)
- *MISMO(linear)*: MI kernel-based SVM method using linear setting (the exponent

Table 3.5: Accuracy of Support Vector Machine Approaches

Dataset	MISMO (RBF)	MISMO (Poly)	MISMO (minimax)	MISMO (linear)	MISVM (RBF)	MISVM (linear)
Musk1	87.04±12.00	82.69±12.79	90.16± 9.49	78.33±12.34 ●	89.21± 9.84	73.87±14.32 ●
Musk2	84.90±10.69	85.76±11.20	84.76±10.05	83.18±11.25	83.92±10.36	70.74±13.14 ●
Mutagenesis-a	70.96± 8.36	75.64±10.13 ○	71.62±10.82	70.26±10.39	66.54± 1.85	66.54± 1.85
Mutagenesis-b	82.36± 8.85	82.46± 8.62	71.28± 6.88 ●	81.06± 9.22	66.54± 1.85 ●	66.54± 1.85 ●
Mutagenesis-c	85.22± 8.35	82.87± 8.22	74.58± 8.95 ●	83.00± 8.29	66.54± 1.85 ●	66.54± 1.85 ●
Suramin	61.18±39.62	67.06±37.44	61.18±40.37	67.06±37.44	67.06±37.44	67.06±37.44
Elephant	82.35± 8.63	83.65± 8.07	85.30± 7.71	81.55± 7.90	75.80± 8.16	79.45± 9.32
Fox	55.20±10.02	56.60±11.52	57.60±11.31	54.75±11.02	49.35± 5.71	49.25± 8.74
Tiger	80.25± 8.60	83.50± 7.50	75.85± 9.02	79.00± 8.13	75.65± 8.43	80.70± 7.42
Trx	86.08± 3.83	86.70± 5.54	87.06± 2.29	87.06± 2.29	87.06± 2.29	87.06± 2.29
EastWest	74.00±31.37	72.00±32.81	59.50±34.59	70.50±32.64	52.00± 9.85	60.50±21.67
WestEast	74.00±31.37	72.00±32.81	59.50±34.59	70.50±32.64	22.50±27.87 ●	38.00±25.74 ●

○, ● statistically significant improvement or degradation wrt left-most scheme

was set to 1)

- *MISVM(RBF)*: *MISVM* method using standard single-instance RBF kernel (the standard RBF kernel parameter gamma was set to 0.5)
- *MISVM(linear)*: *MISVM* method using standard linear SVM setting

Note that two different gamma values for RBF kernel were used in *MISMO(RBF)* and *MISVM(RBF)* schemes as we observed that using these two parameter values (0.1 for *MISMO(RBF)* and 0.5 for *MISVM(RBF)*) obtained a better result in each case.

In general, the MI kernel-based approach performs better than the *MISVM* approach. Note that the MI kernel-based approach actually discards the standard MI assumption and follows the collective assumption because all instances are given equal weight in the classification, whereas the *MISVM* attempts to exploit the standard MI assumption. The good performance of the MI kernel approach demonstrates that these MI problems can be properly separated using the MI kernel based on the collective assumption.

Again, we observe that the results for the MI kernel-based methods on the EastWest and the WestEast datasets are the same whereas the results for the *MISVM* (which follows the standard MI assumption) differ greatly (60.5% and 38.0% respectively in the linear case). This conforms to what we have discussed in the previous section.

3.3.3 Results for Distance-based Approaches

In this section, we compare the three distance-based methods described in Section 2.3. They are *OptimalBall*, *CitationKNN* and the nearest distribution method *MINND*. Fol-

Table 3.6: Accuracy of Distance-based Approaches

Dataset	AdaBoost.M1 with OptimalBall (50)	AdaBoost.M1 with OptimalBall (10)	AdaBoost.M1 with OptimalBall normalized (10)	CitationKNN R=2, C=4	MINND K=5
Musk1	87.50±10.69	83.25±11.93	79.71±13.44 ●	90.37±10.40	75.10±14.12 ●
Musk2	83.18±11.95	84.01±11.03	83.34±11.35	84.61±11.54	72.82±14.93
Mutagenesis-atoms	75.33± 9.74	75.33± 9.74	73.35±10.13	73.19± 9.23	45.63±12.55 ●
Mutagenesis-bonds	75.28± 8.38	75.28± 8.38	75.09± 8.54	75.36± 9.48	31.24± 3.48 ●
Mutagenesis-chains	75.28±10.10	74.69± 9.96	72.07±10.11	74.06± 9.29	41.20±11.20 ●
Suramin	31.18±37.78	31.18±37.78	72.94±34.12	67.06±37.44	67.06±37.44
Elephant	79.55± 8.17	78.55± 9.08	74.20±10.02	50.00± 0.00 ●	74.75±10.55
Fox	48.65± 7.84	48.50± 7.80	55.90±10.45	50.00± 0.00	58.45±10.04 ○
Tiger	66.25±10.67	64.75± 9.62	68.85± 8.16	50.00± 0.00 ●	66.45± 9.96
Trx	84.94± 5.74	85.73± 5.09	89.43± 5.82	87.63± 4.05	87.01± 2.39
EastWest	79.00±24.80	79.50±24.72	73.00±28.80	45.00±24.10 ●	57.00±30.17
WestEast	45.50±29.38	47.00±30.00	50.50±35.17	50.50±27.06	74.50±26.11

○, ● statistically significant improvement or degradation wrt left-most scheme

lowing Auer and Ortner (2004), the *OptimalBall* was applied as a weak learner in a boosting approach. In our experiment, we simply used the *AdaBoost.M1* boosting algorithm implemented in WEKA, with *OptimalBall* as its base learner. The experimental settings were as follows:

- *AdaBoost.M1 with OptimalBall*: Three different parameter settings were used for this approach. They are boosting with 50 iterations, boosting with 10 iterations and boosting with 10 iterations on normalized data (where each attribute was normalized to the [0,1] range).

- *CitationKNN*: In this experiment, we adopted the best experimental setting reported in (Wang & Zucker, 2000) where this scheme was evaluated on the Musk datasets. We set the number of *references* (R) to 2 and the number of *citers* (C) to $R + 2$ which is 4. This setting implies that *citers* are more important than *references* for MI problems. Moreover, the results were obtained by using the *minimal Hausdorff distance* method.

- *MINND*: We set the number of neighbors to 5.

The experimental results on the 12 datasets from Table 3.1 are shown in Table 3.6.

The three *AdaBoost.M1* variants with *OptimalBall* perform similarly, except the scheme that used the normalization option, which exhibited a significant degradation on the Musk1 dataset. Increasing the number of iterations from 10 to 50 in boosting does not show a distinct improvement in this experiment. Comparing the first two schemes in Table 3.6 (*AdaBoost.M1 with OptimalBall 50 iterations* and *Adaboost.M1 with OptimalBall 10 iterations*), we see that the accuracy on five datasets is increased with 50 iterations, kept the same on three and slightly decreased on four. The most obvious improvement occurs on the Musk1 dataset, where accuracy increases from 83.25% to 87.50%. However, this improvement is not significant.

The *CitationKNN* method obtains the best results on the Musk datasets. The accuracy estimates reaches 90.37% on Musk1 and 84.61% on Musk2. However, the *CitationKNN* algorithm seems not suitable for the image retrieval datasets. The accuracy estimates on the three image datasets are all 50% and two of these results are significantly worse compared with the other methods.

The *MINND* method does not perform well in the drug activity prediction domain. Compared with the other methods, four out of the six exhibit significant degradations. On the other hand, the *MINND* method works well on the three image datasets. Especially on the Fox dataset, its accuracy is significantly higher than that of the other methods.

In summary, the *CitationKNN* method seems suitable for the drug activity prediction problems, especially for Musk datasets, but it does not work well on the image retrieval problems. The *MINND* method has some advantage for the image retrieval problem compared with the other two methods but performs worse on the drug activity prediction problem. *AdaBoostM1* with OptimalBall does not always perform best but it is almost always competitive or better than the other two distance-based methods on all MI domains that we tested.

3.3.4 Results for Approaches based on Single-instance Learning

In this section, three different methods of applying single-instance learners to MI problems have been investigated. They are *Wrapper*, *SimpleMI* and *MIBoost*. These methods have been introduced in Section 2.4.

1. Wrapper

For the *Wrapper* method described in Section 2.4.1, we first ran two sets of experiments to investigate the performance of the three different prediction methods and three different weighting schemes respectively, as they are the two key points to obtain good results (Frank & Xu, 2003). Then we ran experiments to compare the performance of various single-instance learners used in conjunction with the *Wrapper*.

Table 3.7: Accuracy of *Wrapper* with Different Prediction Methods (Base Learner: C4.5)

Dataset	Method 1	Method 2	Method 3
Musk1	82.28±12.01	81.88±11.89	72.46±13.07 ●
Musk2	78.44±12.95	78.56±12.78	68.16±12.69 ●
Mutagenesis-atoms	75.94± 9.74	77.10± 9.75	68.88± 4.79 ●
Mutagenesis-bonds	80.76± 8.82	82.35± 8.61	74.43± 6.27 ●
Mutagenesis-chains	83.75± 7.64	83.45± 8.29	74.64± 6.93 ●
Suramin	67.06±37.44	67.06±37.44	67.06±37.44
Elephant	80.25± 9.00	78.55± 9.88	67.40± 6.98 ●
Fox	63.80±10.10	62.60±10.88	56.40± 6.89 ●
Tiger	76.55± 9.09	75.20± 9.07	67.30± 9.08 ●
Trx	87.73± 2.86	87.68± 2.95	19.41± 5.88 ●
EastWest	52.50±33.62	59.00±32.86	51.00±10.00
WestEast	49.50±30.56	53.00±30.83	46.50±22.76

∘, ● statistically significant improvement or degradation wrt left-most scheme

a. *Wrapper* with three different prediction methods

The three different prediction methods are:

- Method 1: arithmetic average of the class probabilities of all individual instances
- Method 2: geometric average of the class probabilities of all individual instances
- Method 3: using the maximum class probability of all individual instances

For this set of experiments, three single-instance classifiers implemented in WEKA were used as base learners. As the first scheme, we used the C4.5 decision tree learner with default settings. As the second scheme, *SMO* using RBF kernels was used. In order to meet the requirement of the *Wrapper* method, that the single-instance learner must be able to provide class probability estimates, we fit logistic models to the output of the support vector machine (Frank & Xu, 2003). Moreover, we standardized the attributes in order to get a better result. As the third scheme, we used linear logistic regression with default settings as the base learner. The experimental results for the three schemes are shown in Tables 3.7, 3.8 and 3.9 respectively.

It is obvious that using the third method, predicting based on the maximum class probability, which actually follows the standard MI assumption, is the worst one among the three. The accuracy estimates for the third prediction method are significantly worse on 9 out of 12 datasets when applying C4.5 as the base learner (see Table 3.7), 4 out of 12 when applying *SMO* (see Table 3.8) and 5 out of 12 when applying linear logistic regression (see Table 3.9). The other two prediction methods, arithmetic average and geometric average, perform similarly for all three base learners. This shows that using the standard MI assumption at prediction time is not suitable, as the *Wrapper* actually

Table 3.8: Accuracy of *Wrapper* with Different Prediction Methods (Base Learner: SMO with RBF kernel)

Dataset	Method 1	Method 2	Method 3	
Musk1	90.68±10.03	90.33± 9.84	84.59±12.06	
Musk2	85.89±10.51	85.63±10.40	79.17±10.70	●
Mutagenesis-atoms	66.54± 1.85	66.54± 1.85	66.54± 1.85	
Mutagenesis-bonds	66.54± 1.85	66.54± 1.85	66.54± 1.85	
Mutagenesis-chains	71.34± 5.15	71.35± 5.92	66.54± 1.85	●
Suramin	32.94±37.44	32.94±37.44	67.06±37.44	
Elephant	82.00± 9.16	82.25± 9.19	74.90± 8.38	●
Fox	62.70±10.28	63.25±10.13	62.40± 7.16	
Tiger	78.55± 7.63	79.10± 7.50	71.55± 7.97	●
Trx	87.06± 2.29	87.06± 2.29	84.77± 5.57	
EastWest	58.50±34.86	57.50±35.80	62.00±21.46	
WestEast	58.50±34.86	57.50±35.80	42.00±18.42	

○, ● statistically significant improvement or degradation wrt left-most scheme

Table 3.9: Accuracy of *Wrapper* with Different Prediction Methods (Base Learner: Linear Logistic Regression)

Dataset	Method 1	Method 2	Method 3	
musk1	80.22±13.08	79.67±13.44	71.22±13.15	●
musk2	82.19±11.76	82.48±12.02	73.06±12.77	●
mutagenesis3-atoms	66.54± 1.85	65.90± 2.46	66.54± 1.85	
mutagenesis3-bonds	67.18± 2.50	67.71± 3.00	66.54± 1.85	
mutagenesis3-chains	71.08± 6.36	71.09± 6.49	66.54± 1.85	●
suramin	67.06±37.44	67.06±37.44	67.06±37.44	
elephant	83.10± 8.61	82.90± 8.50	70.75± 7.83	●
fox	57.75±11.13	57.75±11.13	57.00± 7.14	
tiger	78.60± 8.96	78.15± 8.49	68.35± 7.88	●
Trx	87.06± 2.29	87.06± 2.29	83.28± 6.56	
EastWest	61.50±33.97	60.00±34.08	62.00±21.46	
WestEast	61.50±33.97	60.00±34.08	50.00± 7.11	

○, ● statistically significant improvement or degradation wrt left-most scheme

Table 3.10: Accuracy of *Wrapper* with Different Weighting Schemes (Base Learner: C4.5)

Dataset	$w_i = \frac{\sum N_i}{N \cdot N_i}$	$w_i = \frac{1}{N_i}$	$w_i = 1.0$
Musk1	82.28±12.01	72.31±12.68	75.47±14.99
Musk2	78.44±12.95	72.54±12.64	70.82±13.59
Mutagenesis-atoms	75.94± 9.74	66.54± 1.85 ●	76.10± 8.49
Mutagenesis-bonds	80.76± 8.82	66.54± 1.85 ●	79.81± 8.74
Mutagenesis-chains	83.75± 7.64	66.33± 2.11 ●	82.19± 8.64
Suramin	67.06±37.44	67.06±37.44	67.06±37.44
Elephant	80.25± 9.00	75.75± 9.25	78.95±10.45
Fox	63.80±10.10	61.15± 8.19	62.20±11.02
Tiger	76.55± 9.09	71.70± 9.02	74.45± 9.01
Trx	87.73± 2.86	87.06± 2.29	87.73± 2.86
EastWest	52.50±33.62	67.50±32.86	53.50±35.71
WestEast	49.50±30.56	67.50±32.86	43.50±34.56

○, ● statistically significant improvement or degradation wrt left-most scheme

applies the collective assumption at training time (given each instance in a bag equal weight). It is better to apply the collective assumption consistently at both training and testing time.

b. *Wrapper* with three different weighting schemes

According to (Frank & Xu, 2003), the weighting scheme of the *Wrapper* method is important to get good results on the Musk datasets. In this set of experiments, C4.5 was used as the base learner in the *Wrapper*. The decision tree learner C4.5 is sensitive to instance weights as its splitting stop condition is based on the total weight of the instances at a node. The following three weighting schemes were tested:

1. Give each individual instance a weight of one: $w_i = 1.0$.

2. Give each bag a weight of one and the weights of all individual instances within the bag are set to the same value: $w_i = \frac{1}{N_i}$, where N_i denotes number of instances in the ith bag.

3. Give each bag the same weight so that the total weight of all bags is the same as the total number of instances. The weights of all individual instances within a bag are set to the same value: $w_i = \frac{\sum N_i}{N} \cdot \frac{1}{N_i}$, where N denotes the number of bags.

Table 3.10 shows the results of the three different weighting schemes applied in the *Wrapper* with base learner C4.5. The accuracy estimates were obtained by using the arithmetic average as the prediction method.

It can be seen from the accuracy results listed in Table 3.10 that scheme 3 ($w_i = \frac{\sum N_i}{N} \cdot \frac{1}{N_i}$) is the best among the three. Compared with scheme 3, the performance of scheme 2

($w_i = \frac{1}{N_i}$) is worse on 9 out of 12 datasets, and significantly worse in three cases. For scheme 1 ($w_i = 1.0$), although there was no significant degradation compared with scheme 3, the accuracy estimates decreased on 8 out of 12 datasets.

c. *Wrapper* with different single-instance learners

For these experiments the same single-instance learning schemes as those that have been investigated in (Frank & Xu, 2003) on the Musk datasets were used. One difference is that the accuracy estimates in this thesis were obtained from 100 runs of random splitting (because this enables a more time-grained parallelization of experiments in WEKA) whereas the accuracy estimates shown in (Frank & Xu, 2003) were obtained from 10 runs of stratified 10-fold cross-validation.

A total of 11 single-instance classifiers implemented in WEKA were used. All algorithms were used with their default parameters. However, for the *SMO* classifier, the data was standardized and logistic models were fit to the output for both the linear support vector scheme and the RBF kernel scheme. Note that the arithmetic average prediction method and the third weighting method (i.e. $w_i = \frac{\sum N_i}{N} \cdot \frac{1}{N_i}$) were used in all *Wrapper* schemes. The entire experimental results obtained from all 11 different schemes based on the 12 MI datasets from Table 3.1 are listed in Table 3.11 and Table 3.12.

In order to compare these results and provide a clear view, the number of significant wins and the number of wins for each pair of schemes is shown in Table 3.13. The first row of the table displays the 11 *Wrapper* schemes with their corresponding index numbers (from 1 to 11). In the first column of the table, we use the same index number to represent each scheme (also from 1 to 11). Each row of the table lists the number of wins of a particular scheme compared with the other 10 schemes. The results are based on a total of 12 datasets. Thus, 12 is the highest number and indicates a particular scheme performs better than another scheme on all 12 datasets. For example, the entry with row index *1* and column index *8* is “3 (8)”. It means the first scheme (*AdaBoost.M1 with PART*) outperforms the eighth scheme (*SMO with RBF Kernel*) on 8 datasets with 3 significant wins. The last column of the table shows the total number of significant wins and the total number of wins for each scheme compared with the other 10 schemes.

From Table 3.11, we see that the accuracy obtained from the *Wrapper* with base learners that apply ensemble methods (i.e. the first four schemes listed in Table 3.11) are similar,

Table 3.11: Accuracy of *Wrapper* with Different Base Learners (1)

Dataset	AdaBoost.M1 with PART	AdaBoost.M1 with C4.5	Bagging with PART	Bagging with C4.5	PART	C4.5
Musk1	85.22± 12.71	86.49±11.19	84.57±11.77	84.04±10.96	81.74±14.72	82.28±12.01
Musk2	81.97± 12.23	81.69±12.19	81.13±12.42	79.35±12.23	81.10±11.33	78.44±12.95
Mutagenesis-a	77.31± 8.90	77.26± 9.48	77.37± 9.85	77.27± 8.68	75.67± 9.40	75.94± 9.74
Mutagenesis-b	83.56± 8.14	82.97± 9.11	83.35± 8.24	82.50± 8.28	83.83± 7.93	80.76± 8.82
Mutagenesis-c	84.39± 8.15	84.33± 8.47	84.54± 8.39	84.39± 8.40	84.14± 8.18	83.75± 7.64
Suramin	67.06± 37.44	67.06±37.44	67.06±37.44	67.06±37.44	67.06±37.44	67.06±37.44
Elephant	85.40± 8.25	84.45± 7.85	85.05± 9.20	84.10± 8.71	83.05± 8.58	80.25± 9.00
Fox	63.75± 9.57	62.25± 9.54	67.00± 9.97	65.35±11.20	64.65± 9.72	63.80±10.10
Tiger	80.20± 7.35	80.25± 8.15	81.30± 8.06	80.95± 8.03	76.00± 9.40	76.55± 9.09
Trx	87.84± 2.93	87.84± 3.01	87.06± 2.29	87.73± 2.86	87.06± 2.29	87.73± 2.86
EastWest	53.00± 31.64	56.50±30.69	52.50±32.86	51.00±30.13	58.00±36.74	52.50±33.62
WestEast	53.00± 31.64	57.50±31.28	52.50±29.62	50.50±29.73	47.50±35.80	49.50±30.56

∘, • statistically significant improvement or degradation wrt left-most scheme

Table 3.12: Accuracy of *Wrapper* with Different Base Learners (2)

Dataset	AdaBoost.M1 with PART	SMO linear	SMO RBF	Linear Logistic Regression	IBk	NaiveBayes
Musk1	85.22± 12.71	83.16±12.58	90.68±10.03	80.22±13.08	83.71±11.61	74.46±15.13 •
Musk2	81.97± 12.23	82.52±11.34	85.89±10.51	82.19±11.76	78.16±12.16	76.67±14.33
Mutagenesis-a	77.31± 8.90	66.54± 1.85 •	66.54± 1.85 •	66.54± 1.85 •	81.35± 8.83	66.54± 1.85 •
Mutagenesis-b	83.56± 8.14	66.54± 1.85 •	66.54± 1.85 •	67.18± 2.50 •	84.61± 8.48	64.03± 8.26 •
Mutagenesis-c	84.39± 8.15	68.05± 4.77 •	71.34± 5.15 •	71.08± 6.36 •	84.97± 7.71	60.26±10.65 •
Suramin	67.06± 37.44	32.94±37.44	32.94±37.44	67.06±37.44	78.82±34.83	67.06±37.44
Elephant	85.40± 8.25	82.10± 8.65	82.00± 9.16	83.10± 8.61	79.45± 7.72	81.35± 8.04
Fox	63.75± 9.57	59.95± 9.96	62.70±10.28	57.75±11.13	54.60± 9.63 •	54.80± 6.78 •
Tiger	80.20± 7.35	80.15± 9.22	78.55± 7.63	78.60± 8.96	78.50± 7.44	70.30± 8.43 •
Trx	87.84± 2.93	87.06± 2.29	87.06± 2.29	87.06± 2.29	87.73± 2.86	87.06± 2.29
EastWest	53.00± 31.64	54.50±36.30	58.50±34.86	61.50±33.97	52.00±34.02	59.00±33.62
WestEast	53.00± 31.64	54.50±36.30	58.50±34.86	61.50±33.97	51.50±33.68	59.00±33.62

∘, • statistically significant improvement or degradation wrt left-most scheme

Table 3.13: Comparison of Each Pair of *Wrapper* Schemes: Number of Significant Wins (Number of Wins)

	AdaBoost.M1		Bagging									
	PART	C4.5	PART	C4.5	PART	C4.5	SMO linear	SMO RBF	Linear Logistic	IBk	Naive Bayes	Total
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	
1	-	0 (6)	0 (7)	0 (8)	0 (8)	0 (10)	3 (9)	3 (8)	3 (8)	1 (8)	6 (9)	16 (81)
2	0 (4)	-	0 (5)	0 (7)	0 (8)	0 (10)	3 (11)	3 (7)	3 (8)	0 (8)	6 (9)	15 (77)
3	0 (4)	0 (6)	-	0 (10)	0 (8)	0 (9)	3 (8)	3 (7)	4 (7)	1 (7)	5 (8)	16 (74)
4	0 (2)	0 (4)	0 (1)	-	0 (8)	0 (9)	3 (9)	3 (8)	3 (8)	1 (5)	5 (9)	15 (63)
5	0 (3)	0 (3)	0 (2)	0 (3)	-	0 (6)	3 (7)	3 (6)	3 (5)	1 (4)	4 (8)	14 (47)
6	0 (1)	0 (1)	0 (1)	0 (1)	0 (5)	-	3 (6)	3 (6)	3 (6)	1 (4)	4 (8)	14 (39)
7	0 (3)	0 (1)	0 (3)	0 (3)	0 (4)	0 (6)	-	0 (2)	0 (4)	0 (6)	1 (7)	1 (39)
8	0 (4)	0 (5)	0 (4)	0 (4)	0 (5)	0 (6)	1 (6)	-	1 (4)	0 (7)	5 (7)	7 (52)
9	0 (3)	0 (3)	0 (3)	0 (3)	0 (5)	0 (5)	0 (6)	0 (6)	-	0 (6)	2 (9)	2 (49)
10	0 (4)	0 (4)	0 (5)	0 (6)	0 (8)	0 (7)	4 (6)	4 (5)	3 (6)	-	4 (8)	15 (59)
11	0 (2)	0 (2)	0 (2)	0 (2)	0 (2)	0 (3)	0 (3)	0 (3)	0 (0)	0 (4)	-	0 (23)

and they often outperform the schemes using other simple classifiers. For example, the first and the third schemes listed in the table, applying *AdaBoost.M1* with *PART* and *Bagging* with *PART* as base learners, often perform better than applying the corresponding simple classifier, *PART*: for the first scheme and the third scheme, both have 8 out of 12 accuracy results which are higher than those obtained from the fifth scheme (*Wrapper with PART*), although there is no significant difference. Similarly, for the second and the fourth schemes (applying *AdaBoost.M1* with *C4.5* and *Bagging* with *C4.5*), 10 out of 12 and 9 out of 12 results respectively are higher than those obtained from the sixth scheme (simply applying *Wrapper with C4.5*).

In Table 3.12, we compare the first scheme, applying *AdaBoost.M1 with PART* as the base learner of the *Wrapper*, which is the best scheme overall, with the other five simple classifiers. Compared with applying *AdaBoost.M1 with PART* scheme, applying *SMO with linear model* performs worse on 9 datasets. Applying *SMO with RBF kernel* or *Linear Logistic Regression* performs worse on 8 datasets. And each case results in three significant degradations on the three Mutagenesis datasets. Applying the *one nearest neighbor* method (*IBk*) as the base learner, the accuracy decreases on 8 out of 12 datasets with one significant degradation on the Fox dataset. Accuracy is worst for the last simple scheme, applying *NaiveBayes*. Compared with the first scheme, a total of 9 out of the 12 datasets result in degradations, of which six are significant.

All the above-mentioned comparisons are summarized in Table 3.13 (see the second row with row index *1* of the table). Comparing the first scheme (*AdaBoost.M1 with PART*) with all other 10 schemes on the 12 datasets, a total of 81 wins with 16 significant wins were obtained (shown in the last column of the table).

The experimental results of the 11 schemes on the Musk datasets are similar to those shown in (Frank & Xu, 2003). The highest accuracy on the Musk datasets was obtained by using the support vector machine with RBF kernels, which reaches 90.68% on Musk1 and 85.89% on Musk2 (see Table 3.12).

Overall, if we rank the 11 *Wrapper* schemes according the total number of significant wins and the total number of wins (the numbers listed in the last column of Table 3.13), it is obvious that the first four schemes involving ensemble methods are the best. The tenth scheme (*one-Nearest-Neighbor*) is the second best. The eleventh scheme, using

Table 3.14: Accuracy of *SimpleMI* with Different Transformation Methods (Base Learner: C4.5)

Dataset	Arithmetic	Geometric	Minimax
Musk1	80.59±15.22	77.72±15.34	78.29±12.41
Musk2	75.06±13.66	80.66±12.66	80.79±12.49
Mutagenesis-atoms	79.62± 8.75	70.57± 8.89 ●	74.43±10.04
Mutagenesis-bonds	80.94± 7.85	80.31± 8.85	81.27± 8.15
Mutagenesis-chains	77.37± 9.34	74.98± 8.64	80.22± 8.95
Suramin	34.71±40.11	59.41±41.90	59.41±41.90
Elephant	79.60± 8.43	75.65± 9.01	80.75±10.62
Fox	61.50±10.58	64.70± 9.53	58.00± 9.85
Tiger	75.70± 8.47	68.80± 9.70	72.15± 9.05
Trx	86.39± 3.64	85.44± 4.52	83.04± 6.00
EastWest	93.50±16.90	56.00±34.29 ●	54.00±30.72 ●
WestEast	93.50±16.90	56.00±34.29 ●	54.00±30.72 ●

○, ● statistically significant improvement or degradation wrt left-most scheme

Table 3.15: Accuracy of *SimpleMI* with Different Transformation Methods (Base Learner: SMO with RBF kernel)

Dataset	Arithmetic	Geometric	Minimax
Musk1	89.59±10.72	89.78± 9.75	92.56± 8.18
Musk2	84.69±10.20	79.86±11.40	83.00±11.75
Mutagenesis-atoms	71.19± 9.81	68.96± 9.40	69.43± 9.54
Mutagenesis-bonds	75.21± 8.85	72.03± 9.08	72.48± 9.66
Mutagenesis-chains	79.40± 9.31	71.33± 8.91 ●	71.45± 8.58 ●
Suramin	76.47±34.15	61.18±40.37	61.18±40.37
Elephant	77.25± 9.06	78.25± 9.03	77.70± 9.11
Fox	60.30± 9.97	64.65±10.88	59.90± 8.96
Tiger	73.55± 8.08	73.40± 8.19	72.80± 9.36
Trx	85.23± 4.32	81.65± 7.07	77.14± 8.19 ●
Eastwest	70.50±34.15	62.00±34.90	65.00±33.71
Westeast	70.50±34.15	62.00±34.90	65.50±33.86

○, ● statistically significant improvement or degradation wrt left-most scheme

NaiveBayes as the base learner of the *Wrapper*, performs worst.

2. SimpleMI

For the *SimpleMI* method described in Section 2.4.2, we ran two sets of experiments. In the first set of experiments we compared the three different summarization methods and in the second set of experiments we compare the results for applying different single-instance schemes as the base learners.

a. *SimpleMI* with three different summarization methods:

- Method 1: using the arithmetic per-bag mean of each attribute
- Method 2: using the geometric per-bag mean of each attribute
- Method 3: using the per-bag minimum and maximum each attribute

C4.5 with default settings and *SMO with RBF kernel* were used as the base learners for *SimpleMI* and the experiment was performed on the 12 MI datasets from Table 3.1. The results are shown in Tables 3.14 and 3.15. As in the *Wrapper*, the data was standardized for *SMO* and logistic models were fit to the output.

From Table 3.14, we see that *SimpleMI* using the arithmetic mean as the summarization method and the C4.5 as base learner performs extremely well on the East-West problem. The accuracy obtained from this scheme is much higher than the accuracy of the other two methods. The accuracy reaches 93.50% on both the EastWest and the WestEast dataset while the resulting accuracy for the other two methods is only 56% and 54% respectively.

In most cases, the three different summarization methods for the *SimpleMI* algorithm perform similarly. The arithmetic mean summarization method outperforms the other two methods in a few cases. In Table 3.14, apart from the good performance of the arithmetic mean method on the East-West challenge problem, the geometric mean method results in a significant degradation on the Mutagenesis-atoms dataset compared with the arithmetic mean method. In Table 3.15, compared with the arithmetic mean method, the geometric mean and the minimax method exhibit significant degradation on the Mutagenesis-chains dataset, and the minimax method also exhibit significant degradation on the Trx protein dataset.

b. *SimpleMI* with different single-instance learners
Eleven single-instance classifiers, the same ones as those that were investigated for the *Wrapper* method were used as the base learners for *SimpleMI*. For the summarization method of *SimpleMI*, we used the arithmetic mean option. The entire experimental results obtained from the 11 different schemes based on 12 MI datasets are listed in Table 3.16 and Table 3.17. As for the *Wrapper* method, the number of significant wins and the number of wins for each pair of schemes is shown in Table 3.18.

Similar to the results obtained from the *Wrapper* method, the accuracy obtained from the *SimpleMI* with the four base learners involving ensemble methods are similar (see the first four schemes listed in Table 3.16) and they often outperform the schemes using other simple classifiers (see Table 3.16 and Table 3.17). For example, the first scheme, *SimpleMI with Adaboost.M1(PART)*, outperforms each of the other seven simple classifiers on at least 8 out of the 12 datasets.

Table 3.16: Accuracy of *SimpleMI* with Different Base Learners (1)

Dataset	AdaBoost.M1 with PART	AdaBoost.M1 with C4.5	Bagging with PART	Bagging with C4.5	PART	C4.5
Musk1	78.96± 12.38	83.16±13.24	78.89±13.15	80.18±12.44	75.71±13.23	80.59±15.22
Musk2	78.62± 12.72	79.07±12.76	80.77±11.70	81.41±12.27	75.93±13.52	75.06±13.66
Mutagenesis-a	82.15± 8.30	81.56± 8.25	80.61± 8.01	79.56± 8.55	76.33± 9.72	79.62± 8.75
Mutagenesis-b	84.95± 8.09	84.07± 8.22	84.11± 7.86	84.21± 8.42	83.67± 9.49	80.94± 7.85
Mutagenesis-c	82.76± 9.42	80.37± 8.19	82.26± 8.90	80.95± 9.59	79.17± 9.84	77.37± 9.34
Suramin	35.88± 39.80	53.53±42.11	57.06±41.66	59.41±41.18	43.53±44.18	34.71±40.11
Elephant	83.35± 8.04	84.85± 8.18	85.10± 9.04	83.40± 8.22	81.35± 7.81	79.60± 8.43
Fox	62.50± 10.14	63.30± 9.24	64.70± 9.12	64.90± 9.02	60.80± 9.66	61.50±10.58
Tiger	80.30± 8.64	80.65± 8.55	81.30± 7.71	80.70± 8.07	76.15±10.34	75.70± 8.47
Trx	83.94± 5.69	83.99± 5.99	86.08± 3.60	85.88± 3.71	86.24± 4.35	86.39± 3.64
Eastwest	81.50± 29.86	78.00±29.58	85.50±24.92	85.00±25.13	93.50±16.90	93.50±16.90
Westeast	81.50± 29.86	78.00±29.58	82.00±28.00	79.50±29.38	93.50±16.90	93.50±16.90

○, ● statistically significant improvement or degradation wrt left-most scheme

Table 3.17: Accuracy of *SimpleMI* with Different Base Learners (2)

Dataset	AdaBoost.M1 with PART	SMO Linear	SMO RBF	Linear Logistic Regression	IBk	NaiveBayes
Musk1	78.96± 12.38	79.98±14.02	89.59±10.72 ○	72.38±13.78	84.43±12.30	86.97±10.81
Musk2	78.62± 12.72	79.19±11.93	84.69±10.20	76.44±11.07	78.30±12.12	79.39±12.05
Mutagenesis-a	82.15± 8.30	74.10± 8.72 ●	71.19± 9.81 ●	69.80±10.15 ●	79.10± 8.41	69.83± 9.82 ●
Mutagenesis-b	84.95± 8.09	83.40± 8.52	75.21± 8.85 ●	83.01± 8.00	84.82± 8.46	73.77± 9.29 ●
Mutagenesis-c	82.76± 9.42	79.47± 9.08	79.40± 9.31	78.44± 9.60	84.85± 7.29	78.49± 9.09
Suramin	35.88± 39.80	76.47±34.15 ○	76.47±34.15 ○	76.47±34.15 ○	64.12±41.27	39.41±43.71
Elephant	83.35± 8.04	79.60± 7.97	77.25± 9.06	74.65±10.01 ●	68.20±10.31 ●	80.00± 8.76
Fox	62.50± 10.14	55.10±10.44	60.30± 9.97	54.15± 9.87	58.90± 9.31	54.00± 9.10 ●
Tiger	80.30± 8.64	76.80± 9.25	73.55± 8.08 ●	74.80± 9.37	73.80± 9.43	71.70±10.23 ●
Trx	83.94± 5.69	86.34± 4.26	85.23± 4.32	87.31± 4.37	77.37± 8.26 ●	82.65± 7.29
Eastwest	81.50± 29.86	55.00±34.45 ●	70.50±34.15	53.00±36.11 ●	73.50±32.14	69.00±34.66
Westeast	81.50± 29.86	55.00±34.45 ●	70.50±34.15	53.00±36.11 ●	73.50±32.14	69.00±34.66

○, ● statistically significant improvement or degradation wrt left-most scheme

Table 3.18: Comparison of Each Pair of *SimpleMI* Schemes: Number of Significant Wins (Number of Wins)

	Adaboost.M1		Bagging				SMO	SMO	Linear		Naive	
	PART	C4.5	PART	C4.5	PART	C4.5	linear	RBF	Logistic	IBk	Bayes	Total
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	
1	-	0 (5)	0 (4)	0 (4)	0 (8)	0 (8)	3 (8)	3 (8)	4 (10)	2 (9)	4 (9)	16 (73)
2	0 (7)	-	0 (2)	0 (3)	0 (9)	0 (9)	1 (9)	4 (8)	4 (10)	2 (8)	4 (10)	15 (75)
3	0 (8)	0 (10)	-	0 (7)	0 (9)	1 (8)	4 (9)	4 (9)	5 (10)	3 (8)	4 (11)	21 (89)
4	0 (8)	0 (9)	0 (5)	-	0 (9)	0 (7)	3 (10)	3 (9)	5 (10)	3 (8)	4 (11)	18 (86)
5	0 (4)	0 (3)	0 (3)	0 (3)	-	0 (6)	2 (6)	1 (8)	2 (9)	2 (6)	1 (10)	8 (58)
6	0 (4)	0 (3)	0 (4)	0 (5)	0 (4)	-	2 (6)	1 (8)	3 (7)	2 (7)	2 (7)	10 (55)
7	1 (4)	0 (3)	0 (3)	0 (2)	1 (6)	1 (5)	-	1 (6)	0 (10)	2 (5)	2 (7)	8 (51)
8	2 (4)	0 (4)	1 (3)	1 (3)	2 (4)	1 (4)	1 (5)	-	2 (8)	2 (6)	1 (11)	13 (52)
9	1 (2)	0 (2)	0 (2)	0 (2)	1 (3)	1 (5)	0 (1)	1 (3)	-	1 (4)	1 (5)	6 (29)
10	0 (3)	0 (4)	0 (4)	0 (4)	0 (6)	1 (5)	1 (7)	2 (6)	3 (8)	-	3 (8)	10 (55)
11	0 (3)	0 (2)	0 (1)	0 (1)	1 (2)	0 (5)	0 (5)	0 (1)	1 (7)	1 (4)	-	3 (31)

Table 3.18 summarizes the performance of each scheme compared with the others. According to the total number of significant wins and the total number of wins, listed in the last column of table, it is obvious that the first four schemes involving ensemble methods perform better than the other seven schemes. Among these four schemes, the third one (*SimpleMI with Bagging (PART)*) is the best, which obtains a total of 89 wins with 21 significant wins compared with all other 10 schemes on the 12 datasets. On the other hand, the two schemes, that use *Linear Logistic Regression* and *Naive Bayes* as the base learner of *SimpleMI* respectively, perform worst. The remaining five schemes, namely the fifth (*PART*), the sixth (*C4.5*), the seventh (*SMO with linear setting*), the eighth (*SMO with RBF kernel*) and the tenth (*IBk*) perform similarly.

3. MIBoost

For the *MIBoost* method described in Section 2.4.3, two simple and fast decision tree learners implemented in WEKA were used as the weak learners. The first one is a fast decision tree learner called *REPTree* and the second one is a *DecisionStump*. For the *REPTree*, we used unpruned trees and set the maximum tree depth to three. We ran the experiments with the maximum number of boosting iterations set to 10 and 50 respectively. Table 3.19 shows the experimental results on the 12 datasets.

The results show that, in general, a higher accuracy can be achieved by increasing the number of boosting iterations. From Table 3.19 we see that in most cases, *MIBoost* with a maximum of 50 iterations performs better than *MIBoost* with 10 iterations. The most obvious improvement occurs when using *DecisionStump* as the weak learner of *MIBoost* on the Musk1 dataset. The accuracy increases from 66.22% using 10 iterations to 79.09% using 50 iterations.

Comparing the two weak learners, it is obvious that using *REPTree* often yields better results than using *DecisionStump*. As shown in Table 3.19, except on the two versions of the East-West challenge problem, using *REPTree* always performs better than using *DecisionStump*. In the case with 50 iterations, using *DecisionStump* results in three significant degradations on the Mutagenesis datasets compared to using *REPTree*. In the case with 10 iterations, using *DecisionStump* results in four significant degradations (three are on Mutagenesis datasets and the other one is on the Musk1 dataset).

Table 3.19: Accuracy of *MIBoost*

Dataset	REPTree (50)	DecisionStump (50)	REPTree (10)	DecisionStump (10)
Musk1	84.84±11.55	79.09±13.06	81.32±13.98	66.22±13.53 ●
Musk2	79.50±12.39	78.35±12.80	80.55±11.93	71.72±12.30
Mutagenesis3-atoms	79.26± 9.24	68.50± 6.50 ●	77.31± 9.24	67.61± 5.98 ●
Mutagenesis3-bonds	86.73± 7.31	75.85± 9.35 ●	84.97± 8.30	71.58± 9.13 ●
Mutagenesis3-chains	83.09± 7.70	76.95± 9.14 ●	81.54± 8.28	71.51± 8.45 ●
Suramin	67.06±37.44	67.06±37.44	67.06±37.44	67.06±37.44
Elephant	84.70± 8.46	81.40± 8.41	83.65± 8.31	80.50± 8.27
Fox	66.90± 9.58	63.85± 8.34	65.65± 9.92	61.00± 9.72
Tiger	83.65± 8.07	79.90± 8.96	80.40± 9.39	76.70± 8.02 ●
Trx	87.43± 4.59	87.20± 2.96	87.31± 4.09	87.06± 2.70
Eastwest	55.00±32.18	58.50±38.31	59.00±28.76	68.00±32.97
Westeast	56.50±33.83	58.50±38.31	58.50±30.20	68.00±32.97

○, ● statistically significant improvement or degradation wrt left-most scheme

3.3.5 Results for Two-level Distribution Approach

In this section, we evaluate the performance of the Two-level Distribution (TLD) approach. As mentioned before, the required computing time of the original *TLD* algorithm is expensive because there are four parameters that need to be estimated for each attribute using the optimization procedure. In the experiments, only its simplified version, *TLDSimple*, described in Section 2.5.2 was evaluated.

We compare the *TLDSimple* method with the *SimpleMI with NaiveBayes* scheme shown in Table 3.17 and the *MISMO with minimax kernel* scheme shown in Table 3.5. These three schemes are comparable because all of them extract summary statistics (i.e. "metadata") from bags. In the *SimpleMI with NaiveBayes* scheme, the bag-level metadata is the average mean value of each bag. In the *MISMO with minimax kernel* scheme, the metadata is extracted by combining the minimum and the maximum attribute value of each bag. Table 3.20 shows the comparison of these three methods on the 12 datasets from Table 3.1.

From Table 3.20, we see that *TLDSimple* works well on the Musk datasets (85.29% on Musk1 and 79.69% on Musk2). However, both *SimpleMI with NaiveBayes* and *MISMO with minimax kernel* perform better than *TLDSimple* in most cases. *SimpleMI with NaiveBayes* outperforms *TLDSimple* on 11 datasets with 5 significant improvements and *MISMO with minimax kernel* outperforms *TLDSimple* on 10 datasets with 6 significant improvements.

Table 3.20: Accuracy of *TLDSimple* compared with two Related Methods

Dataset	TLDSimple	SimpleMI NaiveBayes	MISMO minimax
Musk1	85.29±12.30	86.97±10.81	90.16± 9.49
Musk2	79.69±12.25	79.39±12.05	84.76±10.05
Mutagenesis-atoms	38.56±11.90	69.83± 9.82 ∘	71.62±10.82 ∘
Mutagenesis-bonds	45.47±12.52	73.77± 9.29 ∘	71.28± 6.88 ∘
Mutagenesis-chains	43.18± 7.60	78.49± 9.09 ∘	74.58± 8.95 ∘
Suramin	32.94±37.44	39.41±43.71	61.18±40.37
Elephant	74.40± 9.33	80.00± 8.76	85.30± 7.71 ∘
Fox	50.30± 6.51	54.00± 9.10	57.60±11.31
Tiger	61.00±10.87	71.70±10.23 ∘	75.85± 9.02 ∘
Trx	65.41±13.42	82.65± 7.29 ∘	87.06± 2.29 ∘
Eastwest	61.00±28.05	69.00±34.66	59.50±34.59
Westeast	61.00±28.05	69.00±34.66	59.50±34.59

∘, • statistically significant improvement or degradation wrt left-most scheme

3.3.6 Results on Text Categorization Datasets

The experimental methodology for the three text categorization datasets is different from the others. As we mentioned before (in Section 3.2), the pre-specified training and test sets were used for these three datasets. Apart from classification accuracy, we also report the *Area Under Curve* (AUC) measure. The reason is that the class distributions for these three datasets are extremely unbalanced, so the high accuracy does not always represent good performance. In these text categorization problem, balanced data is used for training, but the class distribution for the testing data is extremely unbalanced. In test set, the number of positive bags vs the total number of bags in the Component, Function and Process datasets are 64 vs 2412, 58 vs 4472 and 137 vs 10478 respectively (see Table 3.2). For such unbalanced test data, even if a classifier totally fails to distinguish positive examples from negative ones, the estimated accuracy can still be very high. Therefore, we should use the AUC measure instead to evaluate the performance on these datasets.

AUC is the most frequently used performance measure extracted from a *Receiver Operating Characteristic* (ROC) curve. The ROC curve was original introduced by the signal processing community in order to evaluate the capability of distinguishing informative radar signal from noise (Egan, 1975). The ROC curve is constructed as a two-dimensional measure of performance, in which the X axis is the false positive rate and the Y axis is the true positive rate. The AUC is the value of the area under the curve and the value of AUC is between 1 and 0. If the AUC value is equal to 1, it means perfect classification accuracy has been achieved. Table 3.21 shows the estimated accuracy and AUC on the three text categorization datasets obtained from different schemes.

Table 3.21: Accuracy and AUC on Text Categorization Datasets

Dataset	(1) DD	(2) MILR	(3) MILR -ARITH	(4) MISMO (RBF)	(5) MISMO (minimax)	(6) MISMO (linear)
			Accuracy			
Component	74.75	86.19	82.67	82.30	70.19	70.11
Function	92.20	68.20	91.86	91.41	77.93	89.92
Process	91.40	93.84	89.21	91.25	83.78	90.97
			AUC			
Component	0.76	0.78	0.85	0.85	0.83	0.84
Function	0.78	0.75	0.83	0.83	0.79	0.76
Process	0.84	0.83	0.84	0.81	0.80	0.77

Dataset	(7)CitationKNN (R=2, C=4)	(8) SimpleMI with NaiveBayes	(9) Wrapper with SMO (RBF)	(10) Wrapper with SMO (Poly E=2)	(11) MIBoost with REPTree (50)	(12) TLDSimple
			Accuracy			
Component	63.76	85.82	70.27	78.90	64.76	94.49
Function	71.85	92.58	81.82	87.46	84.19	97.05
Process	80.07	91.03	86.30	87.55	84.12	94.90
			AUC			
Component	0.80	0.80	0.87	0.85	0.85	0.79
Function	0.77	0.78	0.88	0.85	0.84	0.80
Process	0.78	0.80	0.83	0.84	0.84	0.81

It is obvious that among the 12 schemes tested, the ninth scheme *Wrapper with SMO(RBF)* performs best. The AUC value on the Component and Function dataset reaches 0.87 and 0.88 respectively, and both are higher than the AUC of any other scheme. On the Process dataset, the AUC value is 0.83 which is the second highest value (0.84 is the highest value). The tenth scheme *Wrapper with SMO(Poly)* appears to be the second best scheme. The AUC on the three datasets is 0.85, 0.85 and 0.84 respectively.

3.3.7 Suitable MI Approaches for Different Application Domains

In order to get a clear view of the different MI approaches' performance on each application domain, the best and the second best scheme for each of the 15 MI datasets is summarized in Table 3.22. We used two different performance measures to determine the ranking. Normally, the ranking is based on the accuracy estimates. But for the three text categorization datasets, the ranking is based on the *Area Under Curve* (AUC) value. The reason for this is explained in Section 3.3.6.

The simple *Wrapper* approach with single-instance learning algorithms can learn accurate models in many cases. From Table 3.22, we can see that on 8 out of the 15 datasets we can achieve the best performance using the *Wrapper* with a suitable single-instance learning algorithm. Among the remaining 7 datasets, we can obtain the best performance on three of them by using *SimpleMI* with a suitable single-instance learning algorithm

Table 3.22: The Best and the Second Best Scheme for the 15 MI datasets

Dataset	Accuracy or AUC	Best Scheme	Accuracy or AUC	Second Best Scheme
Musk1	90.68%	Wrapper with SMO(RBF)	90.37%	Citation KNN
Musk2	85.89%	Wrapper with SMO(RBF)	85.76%	MISMO (Poly)
Mutagenesis-a	82.15%	SimpleMI with AdaBoost.M1(PART)	81.56%	SimpleMI with AdaBoost.M1(C4.5)
Mutagenesis-b	86.73%	MIBoost with REPTree(50)	84.95%	SimpleMI with AdaBoost.M1(PART)
Mutagenesis-c	85.22%	MISMO (RBF)	84.97%	Wrapper with IBK
Suramin	78.82%	Wrapper with IBk	76.47%	SimpleMI with SMO (Linear) SimpleMI with SMO (RBF) SimpleMI with Linear Logistic
Elephant	85.40%	Wrapper with AdaBoost.M1(PART)	85.30%	MISMO with minimax
Fox	67.00%	Wrapper with Bagging (PART)	66.90%	MIBoost with REPTree (50)
Tiger	83.65%	MIBoost with REPTree (50)	83.50%	MISMO (Poly)
Trx	89.99%	DD	87.92%	EMDD
EastWest	93.50%	SimpleMI with PART (or C4.5)	85.50%	SimpleMI with Bagging (PART)
WestEast	93.50%	SimpleMI with PART (or C4.5)	82.00%	SimpleMI with Bagging (PART)
Component	0.87	Wrapper with SMO (RBF)	0.85	MILRARITH MISMO(RBF) Wrapper with SMO(Poly) MIBoost with REPTree (50)
Function	0.88	Wrapper with SMO(RBF)	0.85	Wrapper with SMO(Poly)
Process	0.84	DD MILRARITH Wrapper with SMO(Poly) MIBoost with REPTree(50)	0.83	Wrapper with SMO(RBF) MILR

and on another two by using *MIBoost with REPTree.* Hence it appears that standard supervised single-instance learning algorithms can properly deal with many MI problems if they are wrapped into an appropriate meta learner. This offers a very simple and efficient solution to MI learning.

The most suitable MI approach for each MI domain is different. From Table 3.22, we see that for the drug activity prediction applications, the *Wrapper with SMO(RBF)*, *Wrapper with IBk* and *SimpleMI with AdaBoost.M1 (PART)* deliver very good results. For the six datasets regarding this application domain that were tested, using one of these three schemes can achieve either the best or the second best result. For the image retrieval application, it appears that ensemble techniques may have some advantage in this area. The three best schemes for the three image datasets are different, but they all involve either *boosting* or *bagging.* The Trx protein identification problem is the big exception in the collection of datasets. Here, applying MI algorithms that exploit the standard asymmetric MI assumption can get the best results. The experiments show that using *DD* and *EMDD* delivers the two best solutions for this problem. For the East-West challenge problem, the best scheme is *SimpleMI with PART(or C4.5).* For the test categorization application, the two best schemes are the *Wrapper with SMO(RBF)* and the *Wrapper with SMO(Poly).* This is not surprising as support vector machines are known to perform well on text categorization problems.

From the experimental results, it appears that the methods which attempt to exploit the standard MI assumption do not have any significant advantages. Among a total of 16 MI algorithms that were described in Chapter 2, five algorithms, *DD*, *EMDD*, *MILR*, *OptimalBall* and *MISVM*, make use of the standard MI assumption. All other algorithms ignore this assumption and treat all instances in a bag equally. Our experimental results show that on most datasets we can achieve good results using algorithms that actually discard the standard MI assumption (see Table 3.22). Recall that for the *Wrapper* method, following the standard MI assumption when making predictions actually performs worse than following the collective assumption (this was demonstrated in the first part of Section 3.3.4). The only exception (i.e. where the standard asymmetric MI assumption appears to help) is the Trx protein identification problem, for which *DD* and *EMDD* are the two best classifiers. This means the standard MI assumption may be more suitable for the Trx data than for the other datasets.

It is hard to say which single algorithm performs best on all the different MI problems that were tested. For example, the *Wrapper with SMO(RBF)* performs best on the two Musk datasets but performs badly on the three Mutagenesis datasets, the Suramin dataset and the East-West problem (see Table 3.12). The *Wrapper with IBk* performs quite well on the three Mutagenesis datasets and the Suramin dataset, but it performs badly on the Fox dataset and the East-West problem (also see Table 3.12). This indicates that the MI problems that were investigated have various different aspects independent of their multi-instance nature. Hence the lesson is that researchers may need to put more effort into trying many different methods when addressing a new MI problem. And simple methods based on standard single-instance learners should be tried first.

Chapter 4
Conclusions

Multi-instance learning has become a popular research topic in the machine learning area since 1997 and many MI algorithms have been proposed since then. However, due to the complexity and the diversity of real-world MI problems, current MI research is still in a growth period and not in a fully developed stage. Motivated by the current situation in MI learning, this thesis presents an empirical study of MI algorithms. We empirically evaluated the performance of sixteen MI algorithms. A total of fifteen datasets covering five different real-world application domains have been investigated.

The empirical results in this thesis show that applying standard single-instance learning algorithms to deal with MI problems can yield good classification performance in most cases. This provides a very simple and efficient solution to MI learning problems. Ray and Craven (2005) drew a similar conclusion from their research and found that ordinary supervised learning algorithms can do well in many MI domains and sometimes are the best algorithms for a task.

The results in this thesis additionally show that when applying single-instance learners to MI problems, using the collective assumption is normally superior to using the standard MI asymmetric assumption. For the single-instance learning algorithms evaluated in (Ray & Craven, 2005), the instance-level prediction with the highest confidence was used as the prediction for the bag, which actually follows the standard MI assumption. As a comparison, the average predicted class probability was used in this thesis to determine the bag-level label, which discards the standard MI assumption and follows the collective assumption instead. The results show that using the average prediction method can yield better results for this approach of applying single-instance learners to MI problems.

The results also show that different MI approaches are suited best to different application domains. There is no single MI algorithm that can outperform all other algorithms on all application domains that were tested in this thesis. This conclusion is consistent and lends further support to findings from previous research (Ray & Craven, 2005).

Although some inroads have been made in this thesis, identifying the most suitable MI algorithm for a particular application domain is still an open question. Based on the empirical results presented in this thesis, we found that for the Trx protein identification problem, the standard MI assumption appears to be very suitable and using the *DD* algorithm yields the best solution for this problem. The experimental results in (Ray & Craven, 2005) are also consistent with this observation. For all other datasets simpler approaches based on the collective assumption appear to be sufficient or even superior. Apart from these findings, the results also show that ensemble techniques are useful in the image retrieval applications investigated here and the results demonstrate the superiority of support vector machines on MI text categorization problems.

Bibliography

Amar, R. A., Dooly, D. R., Goldman, S. A. & Zhang, Q. (2001). Multiple-instance learning of real-valued data. In *Proceedings of the 18th International Conference on Machine Learning* (pp. 3–10). Morgan Kaufmann, San Francisco, CA.

Andrews, S., Tsochantaridis, I. & Hofmann, T. (2002). Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press.

Auer, P. (1997). On learning from multi-instance examples: Empirical evaluation of a theoretical approach. In *Proceedings of the 14th International Conference on Machine Learning* (pp. 21–29). Morgan Kaufmann.

Auer, P., Long, P. M. & Srinivasan, A. (1997). Approximating hyper-rectangles: learning and pseudo-random sets. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computation*. ACM.

Auer, P. & Ortner, R. (2004). A boosting approach to multiple instance learning. In *Proceedings of the 15th European Conference on Machine Learning* (pp. 63–74). Berlin Heidelberg: Springer-Verlag.

Blaschke, C., Leon, E. A., Krallinger, M. & Valencia, A. (2005). Evaluation of BioCreAtIvE assessment of task 2. *BMC Bioinformatics*, *6(Suppl. 1)*.

Braddock, P. S., Hu, D. E., Fan, T. P., Stratford, I. J., Harris, A. L. & Bicknell, R. (1994). A structure-activity analysis of the growth factor and angiogenic activity of basic fibroblast growth factor by suramin and related polyanions. *Br. J. Cancer*, *69(5)*, 890–898.

Bradley, A. P. (1997). Use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30(7)*, 1145–1159.

Chevaleyre, Y. & Zucker, J.-D. (2001). A framework for learning rules from multiple instance data. In *Proceedings of the 12th European Conference on Machine Learning.* Springer-Verlag.

Cover, T. M. & Thomas, J. A. (1991). *Elements of Information Theory.* New York: Wiley.

Dietterich, T. G., Lathrop, R. H. & Lozano-Perez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, *89(1-2)*, 31–71.

Edgar, G. (1990). *Measure, topology, and fractal geometry* (2 Ed.). Springer-Verlag.

Egan, J. P. (1975). *Signal Detection Theory and ROC Analysis.* New York: Academic Press.

Feller, W. (1971). *An introduction to probability theory and its applications, Vol.2* (3 Ed.). New York: Wiley.

Frank, E. & Xu, X. (2003). Applying propositional learning algorithms to multi-instance data. Technical Report 06/03, Department of Computer Science, University of Waikato.

Freund, Y. & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning* (pp. 148–156). San Francisco: Morgan Kauffman.

Friedman, J., Hastie, T. & Tibshirani, R. (2000). Additive logistic regression, a statistical view of boosting (with discussion). *Annals of Statistics*, *28*, 307–337.

Gärtner, T. (2000). Kernel-based feature space transformation in inductive logic programming. MSc thesis, University of Bristol.

Gärtner, T., Flach, P. A., Kowalczyk, A. & Smola, A. J. (2002). Multi-instance kernels. In *Proceedings of the 19th International Conference on Machine Learning* (pp. 179–186). San Francisco, CA: Morgan Kaufmann.

Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz.

Maron, O. (1998). *Learning from ambiguity.* PhD thesis, Massachusetts Institute of Technology, United States.

Maron, O. & Lozano-Perez, T. (1998). A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems 10* (pp. 570–576). Cambridge, MA: MIT Press.

Maron, O. & Ratan, A. L. (1998). Multiple-instance learning for natural scene classification. In *Proceedings of the 15th International Conference on Machine Learning* (pp. 341–349). San Francisco, CA.

Michalski, R. S. & Larson, J. B. (1977). Inductive inference of VL decision rules. Workshop in pattern-Directed Inference Systems. In *SIGART Newsletter 63* (pp. 38–44). ACM.

Nadeau, C. & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, *52(3)*, 239–281.

O'Hagan, A. (1994). Bayesian inference. *Kendall's Advanced Theory of Statistics*, *2B*.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In *Kernel Methods - Support Vector Learning*. MIT press.

Ramon, J. & Raedt, L. D. (2000). Multi instance neural networks. In *Proceedings of the 17th International Conference on Machine Learning, A workshop on "Attribute-Value and Relational Learning: Crossing the Boundaries"*.

Ray, S. & Craven, M. (2005). Supervised versus multiple instance learning: an empirical comparison. In *Proceedings of the 22th International Conference on Machine Learning*. Omnipress.

Ray, S. & Page, D. (2001). Multiple instance regression. In *Proceedings of the 18th International Conference on Machine Learning* (pp. 425–432). Morgan Kaufmann, San Francisco, CA.

Reutemann, P. (2004). Development of a propositionalization toolbox. MSc thesis, Computer Science, University of Freiburg, Germany.

Reutemann, P., Pfahringer, B. & Frank, E. (2004). A toolbox for learning from relational data with propositional and multi-instance learners. In *Proceedings of the 17th Australian Joint Conference on Artificial Intelligence* (pp. 1017–1023). Cairns, Australia: Springer-Verlag.

Ruffo, G. (2000). *Learning single and multiple instance decision trees for computer securtity applications*. Doctoral Dissertation, Department of Computer Science, University of Turin, Italy.

Srinivasan, A., Muggleton, S., King, R. & Sternberg, M. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the 4th International Inductive Logic Programming Workshop*, Volume 237 of *GMD-Studien* (pp. 217–232).

Tao, Q., Scott, S. D. & Vinodchandran, N. V. (2004). SVM based generalized multiple-instance learning via approximate box counting. In *Proceedings of the 21th International Conference on Machine Learning* (pp. 779–806). San Francisco, CA: Morgan Kaufmann.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* New York: Springer Verlag.

Wang, C., Scott, S. D., Zhang, J., Tao, Q., Fomenko, D. & Gladyshev, V. (2004). A study in modeling low-conservation protein superfamilies. Technical Report TR-UNL-CSE-2004-0003, Department of Computer Science, University of Nebraska.

Wang, J. & Zucker, J.-D. (2000). Solving the multiple-instance problem: a lazy learning approach. In *Proceedings of the 17th International Conference on Machine Learning* (pp. 1119–1134). San Francisco, CA: Morgan Kaufmann.

Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2 Ed.). San Francisco, CA: Morgan Kaufmann.

Xu, X. (2001). A nearest distribution approach to multiple-instance learning. Technical report, Department of Computer Science, University of Waikato.

Xu, X. (2003). Statistical learning in multiple instance problems. MSc thesis, Department of Computer Science, University of Waikato.

Xu, X. & Frank, E. (2004). Logistic regression and boosting for labeled bags of instances. In *Proceedings of the Pacific Asia Conference on Knowledge Discovery and Data Mining* (pp. 779–806). Springer-Verlag.

Zhang, Q. & Goldman, S. (2002). EMDD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Zhang, Q., Yu, W., Goldman, S. & Fritts, J. (2002). Content-based image retrieval using multiple-instance learning. In *Proceedings of the 19th International Conference on Machine Learning* (pp. 682–689). Morgan Kaufmann.

Zhou, Z. & Zhang, M. (2003). Ensembles of multi-instance learners. In *Prodeedings of 14th European Conference on Machine Learning* (pp. 492–501). Springer.