Learning Instance Weights in Multi-Instance Learning

James Foulds

This thesis is submitted in partial fulfillment of

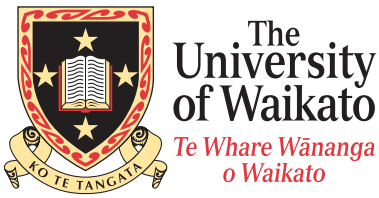
the requirements for the degree of

Master of Science

at the

University of Waikato.

Department of Computer Science



Hamilton, New Zealand

February 2007 - February 2008

Abstract

Multi-instance (MI) learning is a variant of supervised machine learning, where each learning example contains a bag of instances instead of just a single feature vector. MI learning has applications in areas such as drug activity prediction, fruit disease management and image classification.

This thesis investigates the case where each instance has a weight value determining the level of influence that it has on its bag's class label. This is a more general assumption than most existing approaches use, and thus is more widely applicable. The challenge is to accurately estimate these weights in order to make predictions at the bag level.

An existing approach known as MILES is retroactively identified as an algorithm that uses instance weights for MI learning, and is evaluated using a variety of base learners on benchmark problems. New algorithms for learning instance weights for MI learning are also proposed and rigorously evaluated on both artificial and real-world datasets. The new algorithms are shown to achieve better root mean squared error rates than existing approaches on artificial data generated according to the algorithms' underlying assumptions. Experimental results also demonstrate that the new algorithms are competitive with existing approaches on real-world problems.

Acknowledgments

I have been incredibly lucky to live and work in a very supportive environment, which has greatly aided me in the creation of this thesis. I am grateful for the efforts of many people, and the encouragement from many others, without which this thesis would have been almost impossible to complete.

Like all people who write a research thesis, I would like to thank my supervisor. However, a greater level of thanks is due than the typical obligatory mention in the Acknowledgments section. Dr Eibe Frank has repeatedly gone far beyond the call of duty in supporting me in this work. He has always been there to offer clear and insightful advice and constructive criticism, answering my emails promptly and thoroughly even when on leave. This thesis has been greatly improved by his suggestions and observations at all levels of detail. Eibe's depth of knowledge and clarity of thought is an inspiration.

The University of Waikato has provided a very good environment in which to complete a Masters thesis in machine learning. The Department is full of great and friendly minds, and the Machine Learning Lab is almost everything one could want in a work environment — if only it could somehow sprout windows on its walls! The sudden return of natural light would probably prove too much for our sun-deprived bodies however, so perhaps this is for the best. I am grateful to both the University and the Department for financial aid in the form of scholarships.

I thank Cathy Legg and Tim Stokes for giving me teaching assistance jobs, and interesting conversations on a variety of topics. Thanks to Peter Reutemann for assistance with WEKA (even on a public holiday weekend!), Dale Fletcher for helping with access to computational resources, and Dr Mike Mayo for providing datasets and assistance with the image processing parts of my work. Thanks to everyone who has contributed to the WEKA data mining software suite, which has been an essential tool in this work. It is difficult to imagine how I could have completed this research without WEKA.

I would also like to thank my parents, Maureen and Les Foulds, for inspiration

and encouragement. Thanks Mum for the free food and for not charging me rent! Thanks Dad for all the advice, and for helping me with linear programming! Finally, a special "thank you" goes to my partner Erin Bennett, who has been a solid foundation of support all the way through this project.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Multi-instance (MI) learning [Dietterich et al., 1997] is a variant of supervised machine learning that has received a considerable amount of research attention due to both its theoretical interest and its applicability to real-world problems such as drug activity prediction and image classification.

MI learning belongs to the supervised learning paradigm, which aims to solve classification and regression problems by using algorithms to build models of data based on a set of given examples. This thesis is concerned with classification problems, where each example has a classification label that assigns the example into one of a finite number of possible categories. The goal is to "learn" a model based on the training examples that is effective in predicting the classification labels of future examples.

Where MI learning differs from the traditional scenario is in the nature of the learning examples. In the traditional supervised learning scenario, each example is represented by a fixed-length vector of features. All training examples have been (often manually) assigned a class label, which is why the term *supervised learning* is used.

However, in MI learning each example is represented by a multiset (or *bag*, as computer scientists often call it) of feature vectors. In other words, each example contains one or more feature vectors. Classification labels are only provided for entire bags. The task is to learn a model that predicts the classification labels for future bags.

In early MI research, strong assumptions were made regarding the relationship between the feature vectors (known as *instances*) inside the bags and the label of the bag. The restrictive nature of these assumptions was overlooked, mostly due to the ubiquity of the *musk* drug activity prediction datasets [Dietterich et al., 1997], where these assumptions are generally considered to be appropriate. However, in

other problem domains these assumptions do may apply, and more general assumptions may be needed. A significant amount of the more recent research in MI is concerned with cases where more general assumptions hold [Xu, 2003].

This project investigates the case where we may assume that each feature vector instance has a weight value determining the level of influence that it has on its parent bag's class label. This is a more general type of assumption than has been made in most previous approaches, and may be more appropriate for some problem domains. In particular, this type of MI assumption intuitively seems to be a good model for the problem of content-based image retrieval.

1.1 Motivations and Objectives

Before we can apply multi-instance learning algorithms to a problem, we need to consider the relationship between the instances and the classification labels of the bags that contain them. This will determine which type of MI algorithm is appropriate. Because the performance of MI algorithms must necessarily depend upon the interaction between instances and bag labels being compatible with the type of MI concepts that the algorithm is capable of learning, such relationships are known as *MI assumptions* [Xu, 2003].

Although many MI algorithms have been proposed in the literature, most of these algorithms rely upon restrictive assumptions regarding the relationship between the instances in an arbitrary bag and the class label of that bag. Since the year 2000, there has been a trend towards the creation of algorithms that rely upon more general MI assumptions [Xu, 2003]. However, very little research has been done into the case where the instances have a weighted influence on the classification labels of the bags. The purpose of this thesis is to investigate this area, with the intention of better understanding the scenario in order to effectively create and use tools to solve learning problems in domains where this assumption is valid.

A specific motivation for the development of algorithms that make use of this new type of assumption is the problem domain of content-based image retrieval (CBIR), where the task is to identify semantically relevant images in an image library. A common approach to CBIR and other related image mining tasks is to segment the image into multiple smaller regions (see, for example, [Burl et al., 1998],

[Qi et al., 2007] and [Chen et al., 2006]). This makes multiple instance learning a good model for the data — each image can be represented by a bag of feature vectors, where each feature vector represents information extracted from one of the smaller regions (called *segments*) of the image. However, not all segments are created equal, and some segments play a more important role in determining the semantic labels of images than others. For example, when determining whether an image contains a tiger, segments containing green trees are not as important as segments which contain orange and black stripes. It is this type of interaction between instances of varying levels of importance and bag labels that this thesis is concerned with.

Therefore, we have three objectives for this thesis:

1. Identify existing MI learning algorithms that learn weights for instances, and empirically evaluate the effectiveness of such algorithms in a wide variety of problem domains.

2. Precisely formalize MI assumptions where weights are used to model the notion of varying levels of influence that instances have on bag-level class labels.

3. Propose new algorithms that are specifically tailored for learning under these MI assumptions, and rigorously evaluate them on both artificial and real-world datasets.

1.2 Structure of this Thesis

In this chapter we introduced the topic, provided a motivation for the work and concrete objectives for the project. In Chapter 2, we place MI learning in its wider context of machine learning. Different types of machine learning are briefly covered. We then describe MI learning in greater detail. Chapter 3 details related work, including specific details of existing algorithmic approaches to MI learning problems. Chapter 4 presents a detailed study of one such approach that is closely related to the new work in this thesis. Chapter 5 describes new MI assumptions and learning algorithms based on instance weights, and also shows experimental results for these new algorithms on artificial data. The new algorithms are empirically evaluated on real-world problems in Chapter 6. We conclude with Chapter 7, which summarizes the results and presents possibilities for future investigation.

Chapter 2

Background

This chapter gives an overview of machine learning, with emphasis on supervised learning and the multi-instance learning scenario. Other types of learning are described briefly. Multi-instance learning is defined, and the motivations for it are explained. The different types of MI learning are described, but detailed descriptions of the relevant algorithms are not discussed here; see Chapter 3 for these details.

2.1 Machine Learning

Every day, we as humans discover new facts about our world. We interact with the environment around us, and receive feedback through our empirical faculties — our senses. We are able to recognize trends and can begin to anticipate the consequences of our actions. The process that allows us to do this is called learning. It is ubiquitous and most of us take it for granted.

Learning is a task that is normally associated with humans (and intelligent non-human animals), hence the problem of creating machines that can learn falls within the umbrella of artificial intelligence. While the creation of truly "intelligent" machines still seems to be a long way off, machine learning as a practical discipline is a success story of modern artificial intelligence. Many algorithms have been discovered that allow machines to make inferences from observed data, effectively "learning" non-trivial facts and behaviours.

Under the guise of data mining, these algorithms have many commercial applications. Machines are far more efficient and reliable than humans at processing large amounts of data. For this reason, learning algorithms can offer huge cost-saving and efficiency benefits to businesses, and have successful applications in many domains from medicine to marketing.

2.1.1 Supervised Learning

Supervised learning is the branch of machine learning that is concerned with algorithms that can learn concepts from examples. As an input, the algorithm requires a set of example cases, each of which has been given a label corresponding to some important property of the example. The task of the algorithm is to build a model that will generate accurate predictions of the labels of future examples.

With no irony intended, we will now illustrate the concept by way of a rudimentary example. Suppose that we are amateur botanists, and we wish to learn to distinguish between instances of the various species of the *iris* genus of flowering plants. An expert has given us a batch of examples of some of the species of the genus. Once we've seen a few examples of each, we can attempt to infer the defining characteristics of each species. Once we've discovered the pattern, we can become proficient at labeling arbitrary iris plants.

The Iris dataset from the UCI repository [Asuncion and Newman, 2007] is a standard benchmark dataset for the evaluation of machine learning algorithms. It contains data measured from 50 examples of each of three species of iris plants.

The collection of data representing a single example case is referred to as an *instance*. Each instance contains the width and length of the petals of the corresponding example; these data elements are called *features* or *attributes*. Each instance is labeled with the species that the example belongs to; this label is called the *class label*, or sometimes just the *class* of the instance. When measured using a standard algorithm evaluation technique (10-fold cross-validation), the widely-used decision-tree learning algorithm C4.5 is able to learn to predict the species of iris plants in this dataset with an accuracy of approximately 96%.

Having introduced the subject and its terminology via a simple example, we may now formally define the standard supervised machine learning scenario. An instance is a vector of N features concatenated with a class label, of the form $\{x \mid g(x)\}$, where $x = \{x_1, x_2, \ldots, x_N\}$ is the feature vector and $g(x)$ is the label of the instance. Features and class labels are typically either elements of the real numbers (*numeric* attributes) or domain-specific sets of names (*nominal* attributes).

The task is to find $g(x)$, based on a given set of instances. When the class is a nominal attribute, this process is called *classification*. When the class is a numeric

attribute, the process is called *regression*.

The underlying classification process $g(x)$ is known in machine learning terminology as a *concept*. $g(x)$ may be either a function or a non-deterministic process. Given a set of *training* examples to learn from, a supervised machine learning algorithm outputs a model of the data, which is intended to be a best-guess approximation to $g(x)$. Such a model is known as a *concept description*.

This thesis is about a variation of supervised learning called multi-instance learning, which is described in Section 2.2.

2.1.2 Other types of Learning

Unsupervised learning is a type of machine learning where the instances are not given class labels. One form of unsupervised learning, called *clustering*, attempts to split the given dataset into sets of related instances, called *clusters*. "Relatedness" is typically measured via some type of proximity measure, such as the Euclidean distance between the feature vectors.

Semi-supervised learning applies to datasets where some instances are labeled, and some are not. It can therefore be considered to be "in-between" supervised and unsupervised learning. Labeling data is often an expensive manual process, whereas unlabeled data can be acquired automatically and relatively cheaply in some problem domains. Studies have shown [Nigam and Ghani, 2000] that better classification/regression accuracy can be achieved in some problem domains by taking advantage of large sources of unlabeled data.

Reinforcement learning is a style of machine learning where the learning agent receives feedback based on its actions. The agent then attempts to learn a policy of behaviour that is likely to generate the greatest reward at the least cost.

2.2 Multi-Instance Learning

Multi-instance learning [Dietterich et al., 1997], also known as *multiple-instance learning*, or just *MI learning*, is a variation on the standard supervised machine learning scenario. The terms "multi-instance learning", "multiple-instance learning" and "MI learning" will be used interchangeably throughout this thesis.

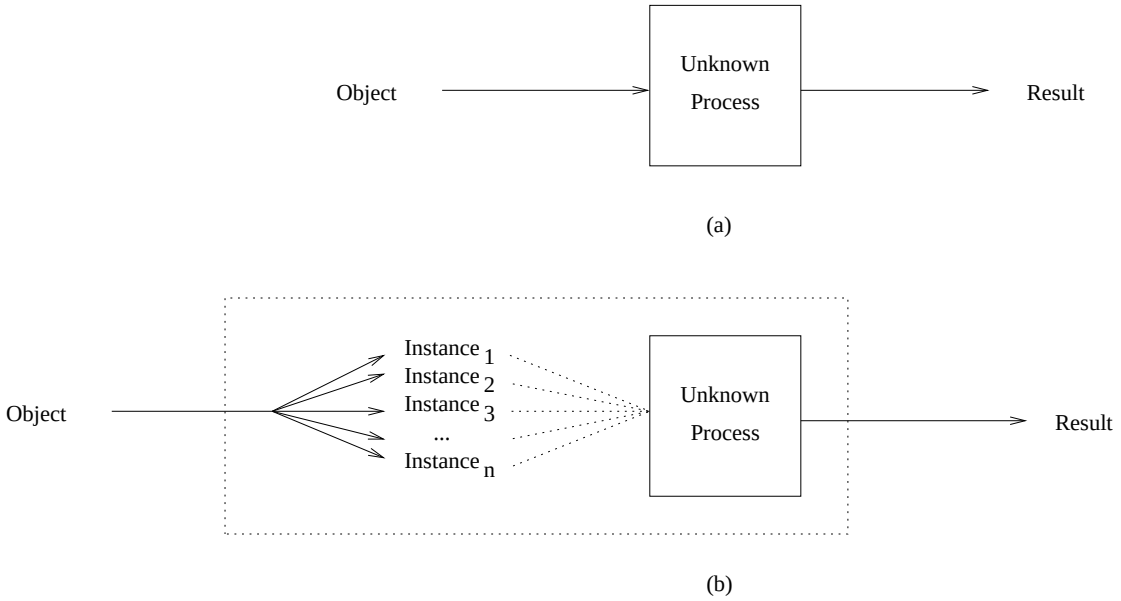


Figure 2.1: (a) The traditional supervised machine learning scenario. (b) Multi-instance learning. Figure based on a similar diagram by [Dietterich et al., 1997].

In standard supervised learning, each example is an instance consisting of a vector of features, augmented with a class label. The task is to learn a function that predicts the class label of an example, given the feature vector.

In MI learning, however, each example consists of a multiset (*bag*) of instances. Each bag has a class label, but the instances themselves are not labeled. The learning problem is to build a model based on given example bags that can accurately predict the class labels of future bags. The difference between standard supervised learning and multi-instance learning is illustrated in Figure 2.1.

An example will once again help to illuminate the concept. This example, called the *simple jailer problem*, is due to [Chevaleyre and Zucker, 2001]. Imagine that there is a locked door, and we have N keychains, each containing a bunch of keys. If a keychain (i.e. bag) contains a key (i.e. instance) that can unlock the door, that keychain is considered to be *useful*. The learning problem is to build a model that can predict whether a given keychain is useful or not.

2.2.1 Definition of Multi-Instance Learning

We now present a formal definition of the MI problem. This formalization is a refinement of those used by [Weidmann et al., 2003] and [Gärtner et al., 2002]. In this thesis we follow the trend established by virtually all work in this field (a notable

exception being [Zhou and Zhang, 2006]) and assume a binary class attribute $\Omega = \{+, -\}$. Let χ be the instance space. Then an MI concept is a function $\nu_{MI} : \mathbb{N}^{\chi} \rightarrow \Omega$. The task in MI learning is to learn this function, based on a number of example elements of the function.

Here, $\mathbb{N}^{\chi}$ refers to the set of all functions from χ to $\mathbb{N}$, which is isomorphic to the set of all multi-subsets of χ, viewing the output of $f(x) \in \mathbb{N}^{\chi}$ as the number of occurrences of x in the multiset. Such functions are known as *multiplicity functions*, and are a direct generalization of *indicator functions* for ordinary sets.

Note that this differs slightly from the formulation used by Weidmann et al., who define an MI concept as a function $\nu_{MI} : 2^{\chi} \rightarrow \Omega$. Here, 2^{χ}, the set of indicator functions over χ, is isomorphic to the power set of χ, but this does not take into account the fact that duplicate instances are allowed in a bag. Our alternative definition of an MI concept explicitly defines the problem examples as multisets rather than just sets. This is important for generalized MI concepts (described later in Section 2.2.4).

2.2.2 Motivations and Applications

MI representations are applicable when examples can be naturally split into instances describing the aspects of the example. There are many diverse applications for MI learning. This section describes several of these.

Drug Activity Prediction

The original motivating application for MI learning was the drug activity prediction problem [Dietterich et al., 1997]. The task is to predict whether or not a given molecule will bind to a target "binding site" on another molecule. Drug molecules that bind well to the target binding site are desirable because they will have the required drug effects. However, one molecule may take on several different conformations (shapes) by rotating its internal bonds. If a single conformation of the molecule can bind to the target binding site, the molecule is considered to be active.

In the *musk* drug activity problem, active molecules emit a "musky" odor that is detectable by human observers. The learning problem is to predict, based on a set of example molecules, whether a given molecule will emit the odor. Dietterich et

al. formulated MI learning to solve this problem. They represented each molecule as a bag containing the different conformations that the molecule can adopt. They showed that their MI-based approach performed better at predicting musk molecule activity than other approaches.

Later, multi-instance learning was applied by [Scott et al., 2005] to the drug activity prediction problem of identifying antagonist drugs, which must bind to more than one target site in order to be labeled as positive.

Inductive Logic Programming

All Inductive Logic Programming (ILP) problems can be transformed into MI problems [De Raedt, 1998], so this is a rich source of applications for MI learning. ILP is a type of machine learning where instances and background knowledge are represented using logic programming (e.g. the PROLOG programming language). ILP problems are relational and hence can be viewed as relational databases. [Reutemann et al., 2004] show how these can be converted into MI problems by performing database join operations to flatten the database into a single table.

One well-known ILP problem is the prediction of mutagenicity of molecules [Srinivasan et al., 1994]. This problem has been successfully tackled using MI learning. Mutagenic molecules are often carcinogenic and can cause damage to DNA, so the identification of such molecules is an important problem. In the ILP dataset, features are recorded for both the molecules, and the atoms within the molecules. [Chevaleyre and Zucker, 2001] formulated four different MI representations for the problem. In all of their representations, each molecule was represented by a bag, but the nature of the instances in the bags varied slightly between representations. The representations are are follows:

1. Each instance represents an atom of the corresponding compound molecule.

2. Each instance represents two atoms of the corresponding compound molecule.

3. Each instance represents an atom, and also contains the molecule-level features of the corresponding molecule.

4. Each instance represents two atoms, and also contains the molecule-level features of the corresponding molecule.

Later, [Reutemann, 2004] created several alternative multiple instance representations of the mutagenesis problem:

1. Each bag contains the atoms of the corresponding compound molecule (similarly to [Chevaleyre and Zucker, 2001]).

2. Each bag contains all atom-bond pairs (tuples) of the corresponding compound molecule.

3. Each bag contains all adjacent pairs of bonds of the corresponding compound molecule.

A similar drug activity prediction problem that has been traditionally approached using ILP is the suramin problem. Suramin analogues (chemical compounds similar to the drug "suramin") have applications as anti-cancer agents [Braddock et al., 1994]. However, the identification of effective suramin analogues for use as anti-cancer drugs is an expensive process due to the large search space. Machine learning models for predicting drug activity from structure can aid in drug design.

The problem has traditionally been approached using ILP [King et al., 1993]. Molecules are represented using the atomic structure and bond relationships similarly to the mutagenicity problem. [Reutemann, 2004] converted the suramin problem into an MI problem using a table flattening operation in a similar fashion to the mutagenesis dataset.

The East-West problem is another ILP problem that has been transformed into an MI problem. The problem was first proposed by [Larson and Michalski, 1977], and later extended for an ILP contest, the East-West Challenge [Michie et al., 1994]. The task is to predict whether a train is eastbound or westbound, given various properties of the cars in the train such as size, shape and load.

The East-West Challenge version of the dataset is easily converted into an MI problem by flattening the relational data into a single table [Reutemann et al., 2004]. The outcome of this transformation is a representation where each train is represented by a bag, with each instance in the bag describing a car belonging to that train.

Fruit Disease Prediction

Fruit disease prediction fits naturally within the MI framework [Xu, 2003]. After harvesting, fruits are usually stored in batches in a warehouse, with each batch coming from a different orchard. However, during storage batches may exhibit symptoms of a disease. Some batches will be more disease-prone than others, so it is useful to be able to predict which batches are the most likely to be disease-prone.

This can easily be represented as an MI problem. One batch of fruits can be represented as a bag, with each individual fruit being an instance. Features are extracted from the individual fruits using non-destructive measures. Training data is acquired by checking for observable symptoms of disease before shipment. Batches that show evidence of the disease are labeled as "positive", otherwise they are labeled "negative". Xu used this MI representation to predict disease incidence in kiwifruit.

Identifying Thioredoxin-fold Proteins

The task of identifying new proteins is conventionally approached by building hidden Markov models (HMMs) on the primary sequence. However, some superfamilies of proteins such as Thioredoxin-fold (Trx-fold) have very low primary sequence conservation, which makes this approach ineffective [Wang et al., 2004]. Wang et al. found that a multiple-instance learning approach performed better than the traditional HMM approach for identifying Trx-fold proteins. [Tao et al., 2004a] also later applied MI learning techniques to the problem.

The Trx-fold superfamily of proteins plays an important role in cellular regulation, and determining the nature of this superfamily is beneficial to the understanding of redox processes [Wang et al., 2004]. The problem is to determine whether a given protein is a member of the Trx-fold superfamily. Wang et al. modeled the problem via multiple instance learning by first finding the primary sequence motif in each sequence, and extracting a fixed-size window around the motif. They claim that the size of their window, which contains 20 residuals upstream and 180 residuals downstream from the primary sequence motif, is known to be large enough to contain the entire Trx fold.

Wang et al. represented each protein as a bag, with feature vectors extracted from sequences within a window as the in-bag instances. They presented three

different feature extraction techniques, namely *motif-based alignment*, *secondary-based alignment* and α-β *signatures*.

Text Categorization

MI learning also has applications to text categorization, where the task is to assign semantic labels to text documents. A document can be represented as an MI bag: instances are obtained by splitting the document into smaller passages. Features such as word occurrence frequencies can be extracted from each passage to form instances.

[Andrews et al., 2002] applied an MI algorithm to the TREC9 dataset (also known as OHSUMED). The dataset consists of thousands of MEDLINE articles, each annotated with Medical Subject Heading (MeSH) terms. They split the documents into passages for their MI representation using overlapping windows with a maximum length of 50 words. Unfortunately, they did not compare their results to those obtained by single-instance approaches.

[Ray and Craven, 2005] used an MI representation for biomedical text-mining. Recent advances in molecular biology have meant that much has been learnt regarding the functions and properties of genes and proteins. So much literature has been published on these topics that it is difficult for database curators to manually build databases linking specific protein functions with relevant articles.

Gene Ontology (GO) codes representing concepts relating to cellular components, molecular functions and biological processes are used by the biomedical community to annotate proteins. It is useful to determine which literature articles link proteins to GO codes. To this end, an $(article, protein)$ pair can be annotated with a GO code if the article links the protein to the component, function or process described by that GO code.

Ray and Craven represent each biomedical article by an MI bag. The instances in the bag each represent a paragraph of the document that contains both the name of the protein and some evidence of a GO code. They consist of word-occurrence frequencies and some extra statistical measures relating to the strength of the connection between the protein and the GO code. The problem is to predict whether the $(article, protein)$ pair should be annotated with a given GO code.

Image Mining

There are many problems within the field of computer vision that multiple-instance learning is naturally applicable to, including content-based image retrieval, object detection and image classification. From a machine learning perspective, these problems belong to the area of *image mining* [Hsu et al., 2002], which encompasses the intersection of the fields of data mining and image processing. Broadly speaking, the general goal in each of these problems is to learn visual concepts. Image mining problems are a major motivation and application for the work introduced in this thesis.

Although standard supervised learning could be applied directly to learn from global features of images, the task of learning visual concepts lends itself well to an MI representation because the target concepts typically only occupy part of the space of an image. Therefore, it makes sense to split the image into smaller regions (segments) [Burl et al., 1998]. Then an image can be represented as a bag of segments. Each instance in a bag contains features extracted from the corresponding segment, such as colour, texture and shape information. Numerous image segmentation techniques exist that can be used for this task.

Content-based image retrieval is a very useful image mining tool for finding relevant content in large digital libraries and multimedia databases. The goal of CBIR is to find semantically relevant images. This must be achieved using the content of the image without reference to manually assigned metadata tags, which are often unavailable. For example, finding all images containing tigers in a given image library is a typical CBIR task.

Some CBIR methods, such as the method described in [Manjunath and Ma, 1996], require the user to manually specify a set of parameters for a prediction model. For instance, patches of orange in the image could be used to predict the presence of tigers. This can be both technically difficult and time consuming for the user, however. An alternative approach is to use machine learning to infer a predictive model from a set of examples provided by the user.

Object detection (also known as object recognition) is roughly the same problem as CBIR, at least from a machine learning task-oriented perspective, but is differently motivated. There are two types of object detection problems: *specific* object

detection, and *generic* object detection. In specific object detection, the goal is to identify images containing a specific object. Generic object detection, on the other hand, attempts to identify objects that belong to a certain category. In *weakly supervised* object detection, the learning program is not told which segments belong to the background and which segments belong to the target object. Object detection is used by artificial intelligence agents such as robots to effectively process sensory input and attempt to make hypotheses about their environment, while CBIR is a task that is performed at the request of a human user who is interested in finding images that are semantically relevant to their own arbitrary set of criteria. Otherwise, the two problems are identical from a machine learning perspective.

Much work has been done regarding the application of MI learning to CBIR / object detection, including [Maron and Ratan, 1998],[Maron and Lozano-Pérez, 1998], [Zhang et al., 2002], and [Chen et al., 2006]. CBIR problems are also used for benchmarking MI algorithms in [Andrews et al., 2002] and [Ray and Craven, 2005].

Both CBIR and object detection/recognition are types of binary classification problems. There are certain target images that we are interested in, and every other image is an irrelevant negative example. In contrast, image classification (otherwise known as image categorization) is explicitly a multi-class problem. Image classification datasets contain images from several different categories, and the task for the learning program is to correctly label each image according to its category.

Work on image classification using multi-instance learning includes [Chen and Wang, 2004] and [Qi et al., 2007]. Although binary classifiers can easily be extended to solve multi-class problems using techniques such as one-against-all, pairwise classification, or error-correcting output codes [Dietterich and Bakiri, 1995], algorithms that are designed to explicitly solve the multi-class problem may perform better. Specifically, many MI algorithms use asymmetric assumptions for binary-class data (such as the standard MI assumption, described later in Section 2.2.3), which may not work well in a multi-class scenario.

Stock Market Prediction

Investors wish to know whether stocks will increase or decrease in value, in order to decide whether to buy or sell stocks for the optimal return on investment. An accurate predictive model would allow investors to make better trading decisions,

and thus earn more money on the share market.

Traditional supervised machine learning approaches to the problem label individual stocks based on the change in value of the stock after a fixed time T [Maron, 1998]. Here, the length of time T is a task-dependant parameter that could range anywhere between minutes and years. For classification, a stock is labeled as positive if the stock price increased after time T. For regression, the stock is labeled with the actual change in market price.

However, supervised machine learning algorithms have had limited success due to the chaotic nature of the system. Unpredictable occurrences such as world events and (often unfounded) rumours can cause fluctuations that are unrelated to the actual economic strength of the stock. Despite this, some stock price fluctuations are actually due to economically fundamental reasons; these are the types of stock value movements that it is hoped machine learning approaches may be able to predict [Maron, 1998].

To counter the noisy data, Maron proposed a multi-instance learning approach for finding stocks that will increase in value due to fundamental economic factors. At training time, positive bags are generated by collecting a fixed number of stocks that all increased in value by the largest amount, and negative bags are generated by collecting a fixed number of stocks that all decreased in value by the largest amount. One positive and one negative bag are created for each time period T in the training set.

The intention is that the multi-instance representation will allow learning algorithms to handle the ambiguity as to whether a stock's value fluctuation was due to legitimate economic factors, or spurious external reasons. As the bag size increases, it is more likely that a positive bag contains a truly positive instance. [Maron, 1998] believes that although this approach is not a general "magical cure" for learning from noisy data, the noise will vary between timesteps, while the true causes of stock performance will remain constant.

Maron found that an MI algorithm, maxDD, performed better at the stock activity prediction task than the predictor developed by Grantham, Mayo, Van Otterloo & Co. (GMO), a Boston-based investment firm.

No comparison is made with single-instance learning algorithms, however, so it is not clear whether the multi-instance representation actually improves performance

on noisy data as intended, and no other work seems to have been done on using MI learning for this purpose. The artificial bag generation method also does not guarantee that the standard MI assumption (see Section 2.2.3) used by the maxDD algorithm actually holds true.

Robot Landmark Matching

During navigation and mapping, mobile robots need to be able to determine their current location with respect to their internal map. This process is called *localization*. Localization is non-trivial because of inaccuracies in effectors (e.g. wheel slippage) and sensors, and possible errors in the internal map. Errors compound with every movement, causing greater inconsistencies over time during the mapping process. For this reason, especially strong localization methods are required during mapping tasks.

Identification of landmarks is one method for implementing localization. If a robot can identify landmarks, it can then determine its position with respect to those landmarks. Applying machine learning to the problem, the task is to learn whether the robot is near a given landmark. Positive examples are images (or other sensor readings) taken near the landmark, and negative examples are images taken when the robot is not near the landmark.

[Scott et al., 2005] represent this as a multi-instance problem, where each example consists of a bag of points derived from sensor readings. They believe that the multi-instance representation may be better able to represent structure information that is lost when the dataset is flattened into a standard supervised learning representation.

Prediction of Hard Drive Failure

Early warnings of impending hard drive failure are important to end-users, as they provide the user with the opportunity to backup their data. Most modern hard drives contain built-in systems designed to provide such an early warning function. However, because drives that emit such a warning are typically required to be replaced under warranty conditions, it is critical for manufacturers to ensure that a low false-alarm rate is maintained.

[Murray et al., 2005] formulated an MI representation of the problem, and presented a new multiple-instance learning algorithm based on the naive Bayesian classifier (mi-NB) that was designed to work well under the low false-alarm rate requirements of the problem domain without sacrificing computational efficiency.

The hard drives studied by Murray et al. implement Self-Monitoring And Reporting Technology (SMART) in order to attempt to predict impending failure. Time-series data are collected by sampling roughly 60 different performance attributes (known as "SMART attributes"), with samples taken after every two hours of operation. The hardware stores a record of the last 300 samples. The existing SMART failure prediction method implements a rudimentary threshold-based algorithm, which the manufacturers estimate to have a failure-prediction accuracy of only around 3-10%, while maintaining a false-alarm rate of 0.1%.

Murray et al. model the hard drive failure prediction problem via an MI representation. They construct feature vectors, which they refer to as *patterns*, by first performing feature-selection to select a subset of the SMART attributes, and then concatenating the selected features of n consecutive samples. Here, n is a parameter to the model. Each hard drive is represented by a bag of patterns.

Murray et al. (2005) set $n = 1$ in their published experimental results, i.e. each pattern represented a single sample, with up to 300 patterns in each bag. Their feature selection method selected 25 SMART attributes for use in learning. Using this representation, they found that their mi-NB algorithm correctly predicted 34.5% of hard drive failures, with a false-alarm rate of 1%. This was a much higher prediction rate than the existing SMART solution, although the false-alarm rate was relatively high.

They also tested a single-instance support vector machine algorithm on the dataset. Patterns, extracted from the example hard drive SMART records and appended with class labels, were used as learning examples. The support vector machine algorithm achieved a higher failure-prediction rate (50.6%) than the multi-instance algorithm, with zero false alarms, but was far more computationally expensive at training time.

2.2.3 The Standard MI Assumption

Most early work on MI learning, notably including [Dietterich et al., 1997] and [Maron, 1998], makes an assumption regarding the relationship between the instances within a bag and the class label of the bag. Dietterich et al. considered this assumption to be so fundamentally important that they included it as part of their definition of multiple-instance learning. We will follow [Weidmann et al., 2003], and refer to this assumption as the *standard MI assumption*.

The standard MI assumption states that each instance has a hidden class label $c \in \Omega = \{+, -\}$. Under this assumption, an example is positive if and only if one or more of its instances are positive. Thus, the bag-level class label is determined by the disjunction of the instance-level class labels.

Formally, let $X = \{X_1, X_2, \ldots, X_n\} \in \mathbb{N}^\chi$ be a bag containing n instances from feature space χ. Each instance has a class label determined by some process $g : \chi \Rightarrow \Omega$. Let $\nu_S : \mathbb{N}^\chi \Rightarrow \Omega$ be a standard MI concept, and equate "+" with the logical constant ''True", and "−" with the logical constant ''False". Then:

$$\nu_S(X) \Leftrightarrow (g(X_1) \vee g(X_2) \vee \ldots \vee g(X_n))$$

It should be noted that the standard MI assumption is asymmetric: if the positive and negative labels are reversed, the assumption has a different meaning. Therefore, when we apply this assumption, we need to be clear which label should be the positive one, and which should be negative.

The standard MI assumption was adopted because it is believed to be appropriate for the *musk* problem domain. In the *musk* problem, it is assumed that a molecule will emit a musky smell if and only if one of its conformations emits a musky smell, hence the standard MI assumption applies [Dietterich et al., 1997].

2.2.4 Generalized MI

Due (at least in part) to the inclusion of the standard MI assumption as part of Dietterich et al's definition of MI learning, it was initially adopted ubiquitously by the fledgeling MI learning community. In more recent years, there has been a trend towards more generalized assumptions [Xu, 2003].

In generalized MI, the standard MI assumption is dropped. Instead, other in-

teractions between instances and the class labels of bags are possible. While most recent authors have (implicitly or explicitly) abandoned the standard assumption, unfortunately many authors have not precisely stated the new assumptions that they have used [Xu, 2003].

Gradually, the definition of multiple instance learning (as used by the machine learning community) has grown to include generalized MI. Most recent authors in the field have dropped the "generalized" prefix, and refer to generalized approaches as part of the multiple instance framework (see, for example, [Xu, 2003], [Chen et al., 2006], and [Dong, 2006]). In particular, Xu explicitly extends the definition of MI learning to include other assumptions.

We contend that the term "multiple-instance learning" should contrast directly with "single-instance learning", and connotes any type of learning where several instances can be included within a single learning example, regardless of the assumptions used. We argue, therefore, that the term should encompass generalized MI as well as the standard MI scenario.

2.2.5 Multiple Instance vs Multiple Part Problems

Chevaleyre and Zucker make a distinction between Multiple Instance problems (MIP) and Multiple Part problems (MPP). Under the generalized view of MI, we consider both types of problems to be within the multi-instance framework. Unlike Chevaleyre and Zucker, we view MPP as a type of MI problem. Regardless, it is insightful to examine the differences between MIP and MPP.

In Chevaleyre and Zucker's framework, MIP problems are the set of problems where the standard MI assumption holds. In MIP, the multiple instance representation is used to describe ambiguity. Each instance in a bag represents some aspect of the instance taken as a whole, but it is ambiguous as to which instances are responsible for a bag's label. Their definition for MIP is as follows:

> **Definition 1 (MIP) [Chevaleyre and Zucker, 2001]**:
> "The multiple-instance learning problem consists in learning a concept from examples that are represented by sets of instances that describe them, on the linearity hypothesis (i.e. the standard MI assumption)."

An example of an MIP is the *musk* drug activity problem. Here, each molecule

is represented by a set of possible conformations (shapes), each of which describe a configuration of the *entire* molecule. If one configuration is positive (i.e. has a musky smell), then the whole molecule is positive.

On the other hand, MPP problems, according to Chevaleyre and Zucker, are problems in which each bag describes the smaller parts of the example. The standard MI assumption is not required in MPP. Under MPP, the instances do not describe aspects of the entire example as in MIP. MPP multiple-instance representations do not represent ambiguity of the whole. Instead, each instance in a bag represents some smaller part of the example. Chevaleyre and Zucker define MPP as:

> **Definition 2 (MPP) [Chevaleyre and Zucker, 2001]**:
> "The multiple-part learning problem consists in learning a concept from examples that are represented by sets of instances that describe their parts, without the linearity hypothesis."

Strictly speaking, this definition should be slightly modified to describe MPP examples as *multisets* rather than *sets*. Some more recent MPP models, such as threshold-based MI and the collective assumption (see Sections 2.2.7 and 2.2.10, respectively) allow repetition of instances in a bag. Defining the concept in terms of multisets allows these models to fit within the framework, while staying true to the spirit of the original definition.

The generalized jailer problem is an example of an MPP problem. Each bag describes a keychain containing a bunch of keys. There is a locked door which is secured by one or more locks. Positive bags are keychains that contain keys that are compatible with all locks on the door. In this problem, each instance describes a key. The instances do not describe the entire example, but instead each describe a smaller part of the example.

In this thesis, we hold a generalized view of MI, which includes all types of learning where examples are multisets of feature vectors, regardless of the assumptions used. Therefore, we consider both MIP and MPP to be part of the broader class MI. However, when designing algorithms, it is important to consider the differences between MIP and MPP problems. Each type of problem may have different requirements in order to learn successfully in that domain.

2.2.6 Xu's Framework for MI Learning

[Xu, 2003] presented a framework for categorizing MI algorithms with respect to the overall approach and assumptions used. Xu's framework (summarized in Figure 2.2) divides MI learning solutions into *instance-based* approaches and *metadata-based* approaches.

Instance-based approaches make the assumption that instances have hidden class labels. The set of hidden instance-level class labels may or may not correspond to the set of bag-level class labels. Methods in this category typically try to estimate a function that assigns class labels to instances, and then use that function to make a prediction at the bag level. All methods that use the standard MI assumption come under this category. Other instance-based assumptions are possible, however. Some of these will be discussed in the following sections.

Methods that use the metadata approach reply on the assumption that the class labels of bags are determined by some meta-level information that describes the example. Algorithms of this type generally apply a transformation that maps bags into a new single-instance feature space, where the features consist of some metadata extracted from the bag. A single-instance learning algorithm can then be applied to the instances in this space in order to make predictions.

Most metadata algorithms are "data-oriented" in that the metadata is extracted directly from the data. In Xu's hierarchy, the alternative to data-orientation is the model-oriented approach, where some model is built on the data, and the metadata is extracted from this model rather than the original dataset.

Data-oriented approaches are further subdivided into "fixed-metadata" approaches, which assume that the instances in a bag are a fixed set of elements associated with that bag, and "random-metadata" approaches, where the assumption is made that the instances are random samples from an underlying distribution. Under this assumption, each bag is actually defined by an instance-space probability distribution, of which the in-bag instances are merely a sample. Random metadata approaches typically map bags into a single-instance feature space that represents an estimate of the parameters of the bag's probability distribution.

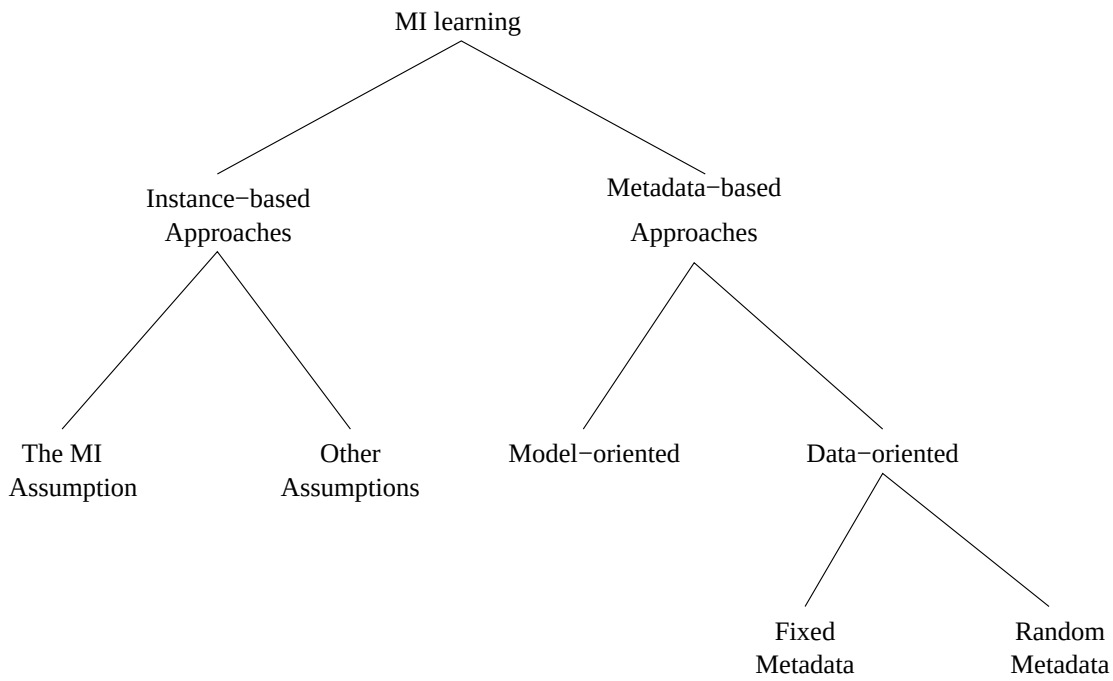


Figure 2.2: Xu's Framework for MI Learning. Based on a similar diagram in [Xu, 2003].

2.2.7 Weidmann's Concept Hierarchy for Instance-Based Generalized MI Learning

[Weidmann et al., 2003] formulated a hierarchy of generalized instance-based assumptions for multi-instance learning. The hierarchy consists of the standard MI assumption and three types of generalized MI assumptions, each more general than the last.

To illustrate the three types of generalized MI assumptions, we will follow [Weidmann, 2003] and use an extended version of Chevaleyre and Zucker's simple jailer problem (discussed earlier in Section 2.2). Recall that in the simple jailer problem, each bag is a keychain containing several keys, and a bag is considered to be *useful* (i.e. positive) if one or more of its keys can unlock a specific door.

Presence-based MI Assumption

In presence-based MI learning, the assumption is that a bag is positive if and only if there exist one or more instances in the bag that belong to a set of required instance-level concepts (i.e. have the required hidden instance-level class labels). This can be visualized as a version of the jailer problem where there are multiple locks on the

door. To unlock the door, we need at least one key that can open each type of lock on the door.

Formally, let $v_{PB} : \mathbb{N}^{\chi} \Rightarrow \Omega$ be a presence-based MI concept, let $\hat{C} \in C$ be the set of required concepts, and let $\Delta : \mathbb{N}^{\chi} \times C \Rightarrow N$ be the function that outputs the count of the number of occurrences of a concept in the bag. Then:

$$v_{PB}(X) \Leftrightarrow \forall c \in \hat{C} : \Delta(X, c) \geq 1$$

It should be noted that the standard MI assumption is a special case of presence-based MI, where $|\hat{C}| = 1$, i.e. there is just one required concept.

Threshold-based MI Assumption

The threshold-based MI assumption states that a bag is positive if and only if there are at least a certain number of instances in the bag that belong to each of the required concepts. Each concept can have a different threshold. In terms of the jailer problem, this is similar to the presence-based MI jailer problem except that multiple copies of each type of lock are allowed, and keys are consumed during the unlocking process. If there are n copies of a certain lock, then we need at least n keys of the appropriate type to unlock it. Anybody who has played the old Microsoft puzzle game *Chip's Challenge* [Microsoft Game Studios, 1990] will be very familiar with this type of problem!

To state the threshold-based assumption formally, let us use the same lexicon as before, and let $v_{TB} : \mathbb{N}^{\chi} \Rightarrow \Omega$ be a threshold based MI concept. Then we have:

$$v_{TB}(X) \Leftrightarrow \forall c_i \in \hat{C} : \Delta(X, c_i) \geq t_i$$

where $t_i \in \mathbb{N}$ is the lower threshold for concept i.

Count-based MI Assumption

Under the count-based MI assumption, there is a maximum and a minimum number of instances from each of the required concepts which must be observed in order for a bag to be positive. Imagine this as the threshold-based jailer problem, except that there is also a stingy jailer who despises wastefulness, and will not allow anybody to open the door if they have too many keys of any particular type.

Formally, let $v_{CB} : \mathbb{N}^\chi \Rightarrow \Omega$ be a count-based MI concept. Then

$$v_{TB}(X) \Leftrightarrow \forall c_i \in \hat{C} : t_i \leq \Delta(X, c_i) \leq z_i$$

where $t_i \in \mathbb{N}$ is a lower threshold for concept i, and $z_i \in \mathbb{N}$ is an upper threshold for concept i.

The Concept Hierarchy

[Weidmann, 2003] showed that these assumptions formed a hierarchy of generality, where *standard MI* $\subset$ *presence-based* $\subset$ *threshold-based* $\subset$ *count-based* (see Figure 2.3 for an illustration).

Therefore, in theory at least, a strong MI learner designed to work under a general assumption should still be able to solve an MI problem where one of the less general assumptions applies. For instance, a strong algorithm designed to use the count-based assumption should work well on a dataset where the generative model is presence-based. See Figure 2.3 for a visual representation of the hierarchy.

2.2.8 The GMIL Assumption

[Scott et al., 2005] introduced a new MI assumption based on theoretical results from geometric pattern recognition. We will refer to this assumption as the *GMIL assumption*. In this model, there is a set of target points $C = \{c_1, c_2, \ldots, c_k\}$. A bag is positive if and only if it contains instances sufficiently close to at least r points, out of the k target points.

Scott et al. extend this model to also include a set of repulsion points $\bar{C} = \{\bar{c_1}, \bar{c_2}, \ldots, \bar{c_{k'}}\}$. In the extended model, for a bag to be positive it may only contain instances that are close to at most s of the repulsion points.

The model can be understood with reference to the *ranked half-Hausdorff* metric using the *weighted infinity norm*. The Hausdorff metric (see, for example, [Edgar, 1990]) provides a measure of distance between two bags of points, and is commonly used in computer vision applications. The sets of target points and repulsion points can be viewed as "ideal bags", where positive bags are within a ranked half-Hausdorff distance of some threshold γ from the ideal positive bag, and at least a ranked half-Hausdorff distance of γ' away from the ideal negative bag.

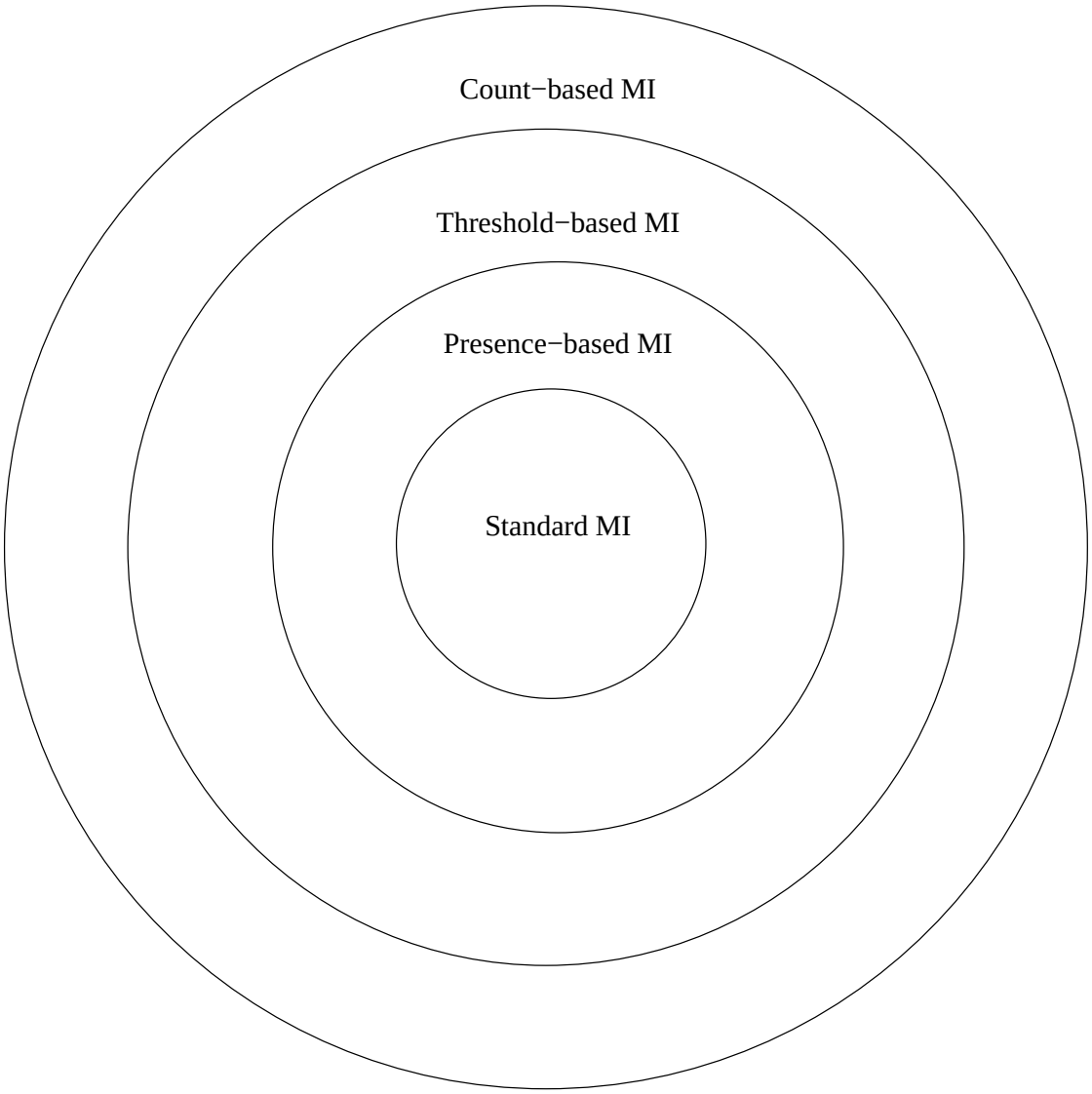


Figure 2.3: Weidmann's hierarchy of instance-based MI concepts.

The Hausdorff distance between bags P and Q is defined to be the largest distance from either a point in P to its closest point in Q, or from a point in Q to its closest point in P, whichever is larger, under some norm. However, this is not robust against noise, so the *ranked* Hausdorff metric is used: instead of using the largest distance, the sth largest distance is used. Scott et al. compute the distance from the bag to the model (i.e. the *half*-Hausdorff metric), but not vice-versa, as it is assumed that the model is accurate and will not contain extraneous points.

Scott et al. used the weighted infinity norm as the instance-level distance measure required to compute the Hausdorff distance. The infinity norm defines the length of a vector (or point) as $\|x\|_\infty = \max(|x_1|, |x_2|, \ldots, |x_n|)$, the largest absolute value of its components. The *weighted* infinity norm allows scaling of the vector components, such as for normalization.

The *ranked half-Hausdorff* metric using the *weighted infinity norm* can be stated formally as

$$\max_{q \in Q}^{s} \left\{ \min_{p \in P} \{ \|p - q\|_\infty \} \right\} ,$$

where $\max^s$ is the sth max, P is a bag, Q is the set of target points, and "$-$" denotes standard vector subtraction. Let a bag P be positive if and only if the above equation evaluates to at most γ. Then a target concept is a set of $k = |Q|$ axis-parallel target boxes, and a bag is positive if and only if it contains points within at least $r = k - s$ of the k target boxes.

To also include a set $\bar{Q}$ of k' axis-parallel repulsion boxes, we must also check that the following formula evaluates to at least γ', which is another constant:

$$\min_{q \in \bar{Q}}^{s'} \left\{ \min_{p \in P} \{ \|p - q\|_\infty \} \right\} .$$

Under this extended model, for a bag to be positive, it must also contain points within at most s' of the k' repulsion boxes.

In terms of Weidmann's hierarchy, Scott et al.'s MI formulation, without repulsion points, is the same as presence-based MI when boxes are viewed as instance-level concepts and the minimum threshold r is equal to the number of target points k. When $r \neq k$, Scott et al.'s model is a more general concept than presence-based MI. Weidmann's threshold and count-based MI concepts generalize presence-based MI

concepts in a different fashion to Scott et al's model, and neither is strictly more general than the other.

Count-based MI concepts can model repulsion points by setting the maximum instances for some instance-level concepts to zero. However, count-based and threshold-based concepts cannot model the case where only r out of k concepts must be present for a bag to be positive. Scott et al's model cannot represent problems where the number of instances belonging to specific constant must be within a given range (as in threshold and count-based MI), as only concept presence rather than concept counts are included in the model.

2.2.9 The DD-SVM / MILES Assumption

The DD-SVM [Chen and Wang, 2004] and MILES [Chen et al., 2006] algorithms also use a generalized MI assumption where bag-level class labels are determined by some relationship to the distance from each of a set of target points. Although the authors of DD-SVM and MILES note that their algorithms do not follow the standard MI assumption, they do not explicitly describe their new assumptions. This section attempts to isolate the common assumptions between these algorithms and thus describe the types of MI concepts that the algorithms attempt to learn.

The DD-SVM / MILES assumption is related to [Scott et al., 2005]'s GMIL assumption, in that distance from a set of target points is used to determine bag labels. However, "distance" is defined differently, and the r-of-k threshold is not used. Instead, a single-instance base learner is used to determine the relationship between proximity to target points and bag-level class labels.

DD-SVM and MILES each define a feature-space mapping where bags are transformed into a single-instance space with attributes representing the closeness of the bag to specific target points in the original instance space. The two algorithms each use a different "closeness" measure and have a different method of selecting target points. As we are currently discussing assumptions rather than algorithms, we will not cover these details here. The reader is referred to Sections 3.3.6 and 3.3.5 for algorithmic details of DD-SVM and MILES, respectively.

The relationship between the "distance from target point" features and bag-level class labels is arbitrary under the DD-SVM / MILES assumption. The assumption

is merely that bag-level labels can be determined entirely from the distance features.

In practice, the algorithms determine this relationship by building a single-instance base learner (typically a support vector machine) on the transformed feature space. By using a powerful enough base learner, Scott et al. type attraction and repulsion points can be modeled under this assumption.

2.2.10 The Collective Assumption

Under the standard MI assumption, only a few special instances (those with a "positive" label) can have any influence on the class label. In contrast, the collective assumption [Xu, 2003] is an MI assumption where all instances in a bag contribute equally to the bag's label.

The collective assumption, designed as a general alternative to the standard MI assumption, was not precisely defined in [Xu, 2003]. However, all of the algorithms presented by Xu that were designed to use this assumption actually depend on the same specific generative model. We will therefore use the term *collective assumption* to refer to this specific model.

The collective assumption is motivated by a probability-theoretical view of the nature of multi-instance bags. Under this view, a bag is not a finite collection of fixed elements (as is generally assumed), but instead is a sample of an underlying population specific to that particular bag. Here, a bag can be modeled as a probability distribution $Pr(X|b)$ over the instance space, where the observed instances were generated by random sampling from that distribution.

Instances are assumed to be assigned class labels according to some (typically unknown) probability function (or nondeterministic probabilistic process) $g(x) = Pr(Y|x)$. Under the collective assumption, the bag-level class probability function is determined by the expected class value of the population of that bag. Let c be a class label $\in Y = \{0, 1\}$, and let b be a bag. Then

$$Pr(c|b) = E_X[Pr(c|x)|b] = \int_X Pr(c|x)Pr(x|b)\,dx\ .$$

To compute this exactly, we must know $Pr(x|b)$, the probability distribution for the bag. However, this is generally not known in practice so we need to use the sample provided by the instances in the bag:

$$Pr(c|b) = \frac{1}{n_b} \sum_{i=1}^{n_b} Pr(c|x_i) \ ,$$

where n_b is the number of instances in the bag. In the limit, as the sample size approaches infinity, the sample version of the equation will approach the population version. Xu developed statistical algorithms for learning this kind of probabilistic concept, and also investigated a simple heuristic algorithm called MIWrapper (see also [Frank and Xu, 2003]). Section 3.3.2 has a detailed description of that algorithm.

2.2.11 Multi-Instance Multi-Label Learning

In traditional supervised learning, *multi-class* learning problems contain more than two classification categories. Each learning instance belongs to exactly one of these categories. An extension to this is *multi-label* learning, where the categories are not mutually exclusive, so that each instance may belong to several class categories [Schapire and Singer, 2000].

[Zhou and Zhang, 2006] formalized multi-instance multi-label learning (MIML), where each multi-instance bag may be associated with multiple class labels. In their formulation, MIML concepts are of the form $f_{MIML} : 2^{\chi} \rightarrow 2^{Y}$, where χ is the instance space, and Y is the set of class categories. Given our interpretation of multi-instance examples as bags (multisets) rather than sets, we modify this definition to be $f_{MIML} : \mathbb{N}^{\chi} \rightarrow 2^{Y}$ (see Section 2.2.1 for more information on this notation).

In MIML, it is clear that the standard MI assumption is not directly applicable, as that assumption is dependent on the learning task being a binary classification problem. Other assumptions regarding the relationships between the instances and the bag-level labels are required for MIML. This is an area that (to the best of our knowledge) has not yet been explored in the literature. Zhou and Zhang did not explicitly state the assumptions used in their work, although their MIMLBoost algorithm uses [Xu and Frank, 2004]'s MI boosting algorithm, and thus implicitly relies on a multi-label version of the collective assumption.

Chapter 3

Related work

Since multi-instance learning was formulated by Dietterich et al. in 1997, many MI learning algorithms have been proposed. In fact, many MI problem domains have motivated entirely new algorithms designed to solve those specific types of problems. This chapter presents an overview of the algorithms in the MI literature.

MI algorithms can be broadly divided into three categories: *purpose-built* algorithms designed specifically to learn MI concepts, *upgraded* single-instance learners that have been modified to learn directly from multi-instance data, and *wrapper* algorithms that convert MI problems into single-instance problems, thereby allowing existing single-instance learners to be applied directly. The approaches explored in this thesis fall mainly into the latter category. The wrapper approaches are more closely related to the new algorithms presented in this thesis, and thus will be described in more detail.

3.1 MI learning using purpose-built algorithms

This section provides a brief description of the main purpose-built MI algorithms. The emphasis will be on background information and high-level descriptions of the algorithms, rather than precise formulations.

3.1.1 APR Formulations

[Dietterich et al., 1997] presented the very first MI learning algorithms, which were designed to solve the MI problem under the standard multiple-instance assumption. Specifically, the algorithms were designed to work well for the *musk* drug activity prediction problem.

The algorithms attempt to build a single axis-parallel hyper-rectangle (APR) that identifies the "positive" region of instance space. A hyper-rectangle is a gener-

alization of a rectangle to an arbitrary number of dimensions. Axis-parallel hyper-rectangles can be viewed as conjunctions of if-then rules on the feature values. At classification time, any bag that contains an instance within the hyper-rectangle is labeled as positive, as per the standard MI assumption.

Dietterich et al. formulated three types of algorithms for learning APR-type standard MI concepts: *basic APR* algorithms, *outside-in* algorithms and *inside-out* algorithms. Basic APR algorithms start by building a rectangle that covers all of the instances in all of the positive bags. This rectangle is likely to contain many false-positives, so various methods are employed to attempt to shrink it while retaining as many instances from positive bags as possible. Outside-in algorithms also start with a large rectangle and attempt to shrink it, but the standard MI assumption is more directly taken into account. These algorithms are designed to ensure that the resulting APR contains at least once instance from each positive bag. Inside-out algorithms start with a rectangle around a single instance, and attempt to "grow" the rectangle outwards.

The most effective APR method on the *musk* data was Iterative Discrimination, an inside-out APR algorithm. This algorithm has three main procedures: *grow*, *discrim* and *expand*. The algorithm starts by *grow*ing a hyper-rectangle over all features with tight bounds on the positive instances. Then it *discrim*inates between the features, selecting a subset of features that best distinguishes between positive and negative instances. The *grow* procedure is repeated, using only the selected features. The algorithm iterates, alternating between growing and discriminating until it converges on a stable hyper-rectangle. However, this creates a very conservative rectangle which does not generalize very well. So another step was added to *expand* the hyper-rectangle in order to increase the probability that new positive instances are contained within the APR.

3.1.2 Diverse Density

The diverse density framework ([Maron, 1998], [Maron and Lozano-Pérez, 1998]) is a probabilistic approach to MI learning under the standard assumption. Diverse density is a measure of the probability that a point in instance space is the positive target point (assuming that there is only one target point) given the bags in the

training dataset. This can be written as:

$$DD(x) = Pr(x = t|B_1^+, \ldots, B_n^+, B_1^-, \ldots, B_m^-),$$

where $x \in \chi$ is a point in instance space, and t is the target concept. The diverse density of a point is computed based on the number of *different* positive bags that have instances close to that point, and how far away the instances from negative bags (which are all assumed to be negative instances according to the standard MI assumption) are from that point. The word "diverse" is used to emphasize the fact that the instances must come from different bags in order to contribute to the measure.

Since a single target point is assumed, the goal of a learning algorithm within the diverse density framework is to find the point with the maximum diverse density. Assuming that the bags are conditionally independent given the target concept, and further assuming uniform prior probabilities for the target, this can be computed as

$$\arg\max_x \prod_{1 \le i \le n} Pr(x = t|B_i^+) \prod_{1 \le j \le m} Pr(x = t|B_j^-).$$

Of course, the probability terms in this equation need to be defined in order to compute this. Maron (1998) proposes two models for estimating the probability that a point is the target concept given a bag. One is the *noisy-or* model, which uses a probabilistic version of the logical *or* operator, and the other is the *most-likely-cause* model, which selects the instance in a positive bag that has the highest probability of being positive.

Maron and Lozano-Perez provide an algorithm (which we will call maxDD) that attempts to find the target concept by maximizing the diverse density over the instance space. As it is not feasible to search the entire instance space for the point with the greatest diverse density, a gradient ascent method is used. Restarts are performed at every instance from a positive bag, because positive instances cause the diverse density peaks, and therefore presumably the target concept is close to some of those instances.

[Zhang and Goldman, 2002] later formulated the algorithm EM-DD, which is a variant on the maxDD algorithm that is based on the expectation-maximization (EM) approach. Although EM-DD uses the same theoretical framework as maxDD,

it has a different method for finding the most likely target point.

The algorithm starts with an initial guess h of the target concept, obtained by selecting a point from a positive bag as in maxDD. It then performs an iterative procedure consisting of an *expectation* step followed by a *maximization* step. The expectation step selects an instance from each positive bag that is most likely to be the cause of that bag's label given the current hypothesis h, using the most-likely-cause estimator from maxDD. Then, the maximization step performs a gradient ascent search based on the selected instances (similarly to maxDD) to find a new h' that maximizes $DD(h')$. The current hypothesis h is reinitialized to h'. The EM loop is repeated until convergence.

EM-DD has a computational efficiency advantage over maxDD. The maxDD algorithm uses a *softmax* function in place of a max function because it is differentiable as required by the gradient ascent search, which increases the computational complexity of the algorithm. Because EM-DD selects only a single instance from each bag during the expectation step, computing the max function becomes trivial and the expensive softmax computation can be avoided. This also causes the algorithm to scale well with increasing bag size.

The authors of EM-DD initially reported improved performance over maxDD on the *musk* data, but this was disputed by later authors. [Andrews et al., 2002] pointed out that the original formulation of EM-DD selected the best hypothesis based on error rate on the test data instead of the training data, and so the reported results were optimistic. They corrected this error, and found that the algorithm's accuracy was not actually superior to maxDD. However, the algorithm is still notable for its improved computational efficiency over maxDD.

3.1.3 ConMIL

[Qi et al., 2007] presented an MI algorithm, Concurrent MIL (ConMIL), that is designed to use concurrency relations between instance-level concepts to solve MIL problems. The algorithm searches for a class probability function $p_i(x)$ over instance space which best represents the concept concurrency relations observed in the training data. Bag level predictions are made using this instance-level probability distribution function according to the standard MI assumption.

The algorithm is motivated by the idea that the concurrency of certain instance-level concepts may be important for MI classification. For instance, in an image classification task, images of beach scenes may be defined as images that contain both *sea* regions and *sand* regions. However, as we will see, ConMIL uses the standard MI assumption for bag-level prediction, and thus fails to make full use of this concurrency relation information.

For the ConMIL algorithm, concurrency relations in the training data are viewed as an order-n tensor, where n is the maximum order of the concurrency relations considered. For example, $\{sand \wedge sea\}$ is an order-2 concurrency relation, while $\{sand \wedge sea \wedge people \wedge sky\}$ is an order-4 concurrency relation. Tensors are abstract mathematical objects that generalize the concepts of scalars, vectors and matrices to higher dimensions. An order-n tensor can be represented as an n-dimensional array. Qi et al. define the *concurrent tensor* T as

$$T_{i_1,i_2,\cdots,i_n} \triangleq p(I_{i_1} \wedge I_{i_2} \wedge \cdots \wedge I_{i_n}) \ ,$$

where I_x ($1 \leq x \leq n_I$ = the total number of instances) is an instance from the training data. Here, $p(I_{i_1} \wedge I_{i_2} \wedge \cdots \wedge I_{i_n})$ represents the probability of the concurrence of n instances $I_{i_1}, I_{i_2}, I_{i_n}$ in the same positive bag. This concurrent probability is computed using multiplication as a fuzzy connective to replace the logical operator "$\wedge$" [Yager, 1980], and a noisy-or model [Maron, 1998].

ConMIL uses a quasi-Newtonian gradient descent search to find the instance-level class probability function $p_i(x)$ that minimizes a least-squares cost function with respect to similarity to the concurrency tensor T. At classification time, the predicted instance-level class probabilities are used to make bag-level predictions according to the standard MI assumption. A *softmax* function is used to make this prediction.

The ConMIL algorithm is related to the work presented in this thesis in that it tries to use information on the relationships between instance-level concepts to make bag-level predictions. However, since it uses the standard MI assumption, these relationships are not actually directly used at prediction time. In this way, it does not directly make use of concept concurrency information to make predictions at all. Concurrency information is used to learn an instance-level posterior class

probability function, but predictions are made using only the posterior probability function, without reference to the concurrency relationships.

For example, suppose in an image classification task a concurrence has been detected between sea regions and sand regions in images of beach scenes. ConMIL uses this concurrence to help compute the probability that a sea region belongs to a beach, and the probability that a sand region belongs to a beach. Now, suppose that the ConMIL classifier is used to make a prediction on an image containing a sea region and a sand region. In order to classify the image, ConMIL uses the predicted probabilities that sea and sand regions each independently belong to a beach scene, but it does not use the probability that an image containing *both* sea and sand regions is a beach scene.

A potential direction of future research would be to try to identify instance-level concepts at classification time, and compare these to the concurrency probabilities stored in the concurrency tensor in order to make predictions. In this way, the concurrency information that ConMIL already computes could be used to make predictions, rather than discarding the information at prediction time in favour of a posterior probability function and the standard MI assumption, which do not make use of concept concurrency relations.

3.1.4 GMIL

[Scott et al., 2005] [1] formulated the GMIL generalized MI assumption, where a bag is positive if and only if it contains instances sufficiently close to at least r of k possible target points, and sufficiently close to at most s' of k' repulsion points. Given the distance measure used, "sufficiently close" is equivalent to being within an appropriately sized axis-parallel box of the target (or repulsion) point. See Section 2.2.8 for more detail on this MI assumption.

Their first algorithm for learning this type of concept was called GMIL-1. The algorithm explicitly enumerates all possible axis-parallel boxes. It creates a single-instance feature space with boolean attributes for each box, signifying whether a bag contains an instance within that box. To reduce the dimensionality of this space, boxes that cover the same instances are grouped together, and only one

[1] Originally published in 2003 as a technical report at the University of Nebraska, Lincoln.

representative box for each group is used.

The training bags are mapped into the feature space, and the single-instance algorithm Winnow [Littlestone, 1987] is trained on the transformed dataset. Winnow is similar to the Perceptron algorithm for learning single-layer neural networks, but it updates its weights multiplicatively instead of additively. This causes a faster convergence rate, in particular when many attributes are irrelevant (such as for GMIL). The Winnow algorithm is used here because (according to Scott et al.) it is known to be able to learn r-of-k type functions.

The task of enumerating all axis-parallel boxes is exponential in the number of dimensions, which makes GMIL-1 very inefficient. GMIL-2 [Tao and Scott, 2004] is an attempt to improve the computational and memory efficiency of the algorithm. The algorithm is roughly the same as GMIL-1, but it selects groups of boxes in a different way. First, GMIL-2 reduces the number of instances to consider by selecting a subset of representative instances, Ψ. Then it constructs groups by considering the boxes represented by the bounding box of each possible subset of Ψ. A breadth-first search approach is used to attempt to efficiently find the sets of groups that are geometrically valid, i.e. all instances within the bounding box of the group are contained within the group.

Although GMIL-2 is far more efficient than GMIL-1, it still suffers from limited scalability [Tao et al., 2004a]. In a further attempt to improve the algorithm's computational complexity, Tao et al. presented a kernel-based reformulation of the GMIL learning problem. The kernel, $k_\wedge$, allows a support vector machine to be applied directly to the problem (see Section 3.2.3). As the computation of $k_\wedge$ belongs to the complexity class *#P-complete* and thus suffers from severe scalability issues that quickly make the problem intractable as the problem size increases, the authors presented a fully-polynomial randomized approximation scheme (FRAPS) for it.

A new kernel based on $k_\wedge$, called k_{min}, was later created in [Tao et al., 2004b]. The k_{min} kernel extends the GMIL model to handle Weidmann's count-based MI concepts (described in Section 2.2.7). Tao et al. found that the k_{min} kernel algorithm performed better than $k_\wedge$ at CBIR tasks and identifying trx-fold proteins.

3.2 Upgraded Single-Instance Learners

A common approach to multiple-instance learning is to upgrade single instance learners to handle the MI learning task. The supervised learning literature provides many single-instance learning algorithms that are well-supported both theoretically and empirically, and these algorithms can provide a solid foundation from which to formulate MI algorithms. Often a single-instance learner can be upgraded to handle a multi-instance task with only minor changes to the algorithm, so this saves on design time as well. Upgraded supervised learning algorithms include k-nearest neighbours, support vector machines, decision trees, logistic regression and boosting.

3.2.1 Nearest Neighbour Approaches

In single-instance supervised learning, the nearest neighbour algorithm makes classification predictions by labeling instances with the class label of the closest instance in the training data (under some norm, such as the Euclidean distance). The algorithm can be made more robust by taking into account not just the nearest instance, but the k-nearest instances. Predictions are made via a majority vote of the k neighbours' class labels. The k-nearest neighbour method is here abbreviated as k-NN.

The most straightforward way to upgrade k-NN to MI learning is to define a norm (distance measure) for MI bags. Then the k-NN algorithm can be applied directly. For this MI norm, [Wang and Zucker, 2000] used the ranked Hausdorff Distance (described earlier in Section 2.2.8). They found that the majority vote method used by standard k-NN often produced sub-optimal results in the MI setting. To improve on this, Wang and Zucker tried two variations on the multi-instance k-NN method.

The first proposed algorithm was Bayesian-KNN, which replaces the majority vote of the k neighbours with a probabilistic method, where the most likely class label is estimated via an application of Bayes' theorem.

Citation-KNN is the other modified nearest neighbour approach proposed by Wang and Zucker. The method is based on a theoretical framework from the field of library and information science, which defines relevance between documents (especially research papers) as related to reference and citation relationships. Under this view, if a research paper cites an earlier work (known as the *reference*), that paper is said to be *related* to the earlier paper. Similarly, a paper that is cited by a

later publication (the *citer*) is considered to be related to the later work.

Wang and Zucker use the citation metaphor to describe nearest neighbour relationships. Under this scheme, the R-nearest neighbours of a bag are viewed as the R-nearest references. To define the C-nearest citers, first consider the neighbour ranking function $Rank(b', b)$. If b is the nth neighbour of b', then $Rank(b', b)$ returns n. The C-nearest citers of b are the C bags that return the lowest neighbour ranking for b.

To classify an instance, Citation-KNN collects the R-nearest references and the C-nearest citers, and returns a positive prediction if and only if there are strictly more positive instances than negative instances in the combined reference/citer collection. Ties are broken in favour of negative predictions because positive bags may contain negative instances that may have erroneously affected the classification in favour of a positive label, whereas the reverse is not possible under the standard MI assumption.

3.2.2 Decision Trees and Decision Rules

[Chevaleyre and Zucker, 2001] presented upgraded MI versions of several standard decision tree and decision rule learning algorithms. Decision tree learners formulate the hypothesis space as a tree of decisions, that typically involve comparing an attribute value with a specified constant. The tree is generally built in a top-down recursive fashion.

When learning a decision tree, the challenge is to determine which attributes to make a decision on, and where to place the *split-point* for that decision. Many modern decision tree learning algorithms, such as ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1993], use *information gain* to select the attributes and split-points. Information is a measure from information theory which roughly represents the amount of information (measured in *bits*) that would be required to determine whether an instance is positive or negative, given that the instance reached that node in the tree. The information gain between a parent node and its child is the information required at the parent, minus the information required at the child [Witten and Frank, 2005].

Information gain is calculated using a concept called *entropy*. Chevaleyre and Zucker formulated a version of entropy for MI data, and were thus able to upgrade

both ID3 and C4.5 into the multi-instance framework with very little extra modification.

They also used a similar strategy for upgrading decision rule learners. The decision rule representation is an alternative to the decision tree, where an arbitrary set of logical rules determines the classification of an instance. Many decision rule algorithms learn rule sets via a *covering* approach, where rules are added incrementally, with each new rule covering some instances that have not yet been covered by the previously added rules. By defining a notion of *coverage* for multi-instance data, Chevaleyre and Zucker were able to upgrade the covering decision rule learning algorithms AQ, CN2 and CHARADE.

3.2.3 Support Vector Machine Approaches

Support vector machine (SVM) algorithms [Cortes and Vapnik, 1995] are a popular approach for single-instance learning. These algorithms attempt to learn a separating hyperplane that divides instance space into positive and negative regions. A hyperplane is a generalization of the notion of a line in Euclidean plane geometry, and a plane in 3-dimensional geometry. A line in Euclidean plane geometry divides the plane into two regions, one on either side of the line. A plane has a similar function in 3-dimensional space. Intuitively, a hyperplane has the same function in an arbitrarily-dimensioned space. The hyperplane learned by a support vector machine provides a decision boundary for classification.

SVM algorithms attempt to learn a *maximum margin* hyperplane, where the hyperplane separates the training instances but is as far away from both the positive and negative training instances as possible. When applied directly to instance-space, the hyperplane learnt by an SVM algorithm is a linear decision boundary. However, this model is not sophisticated enough for some problem domains. For instance, a linear decision boundary cannot represent the exclusive-or (XOR) function from propositional logic. Fortunately, more complex decision boundaries can be represented by hyperplanes in higher-dimensional spaces, where features are derived from combinations of the attributes in the original feature space. Instances are mapped into the higher-dimensional spaces, and the decision boundary in the higher-dimensional space is used for classification.

A further advance was the development of kernel functions for support vector machines. Kernel functions obviate the need to actually perform the mapping into the higher dimensional space (and subsequently more expensive computations in that space) by simulating the inner product of vectors in that space. It turns out that the inner product is the only operation required to build the separating hyperplane, so by computing the kernel function in the original feature space, the hyperplane for the higher-dimensional space can be computed without actually performing the mapping.

To apply the SVM approach to multi-instance learning, [Gärtner et al., 2002] formulated two kernel functions for MI data: the *MI Kernel*, which is based on the set kernel [Haussler, 1999] and the *Minimax Kernel*, which involves mapping bags into a space consisting of minimum and maximum values for each feature.

Later, [Andrews et al., 2002] developed the mi-SVM, which is a reformulation of the support vector machine problem for multi-instance data. The algorithm is designed to learn under the standard MI assumption. The mi-SVM builds a hyperplane over instance space which attempts to include at least one instance from each positive bag in the positive halfspace, and place all instances from negative bags in the negative halfspace, while still maintaining a maximum margin under these constraints.

[Andrews et al., 2002] also formulated the MI-SVM (note the difference in capitalization), which defines a bag-level margin. The algorithm maximizes the margin between the bags of instances and the hyperplane. Andrews et al. believe that the MI problem requires the more fundamental revision of support vector machines that their formulations provide, as opposed to Gärtner et al.'s approach which only modifies the kernel while leaving the SVM formulation otherwise unchanged.

For multi-instance multi-label data (described in Section 2.2.11), [Zhou and Zhang, 2006] use a bag-level constructive clustering approach to convert the dataset into a single-instance multi-label format, and apply the multi-label support vector machine algorithm (MLSVM) to the transformed dataset. This algorithm is called MIMLSVM.

Other approaches using support vector machines for multi-instance learning include DD-SVM [Chen and Wang, 2004] and MILES [Chen et al., 2006]. Both of these algorithms convert MI problems into single-instance problems by mapping

bags into single-instance feature spaces, and then apply support vector machine algorithms to the transformed dataset. Although the authors of these algorithms define them in terms of the support vector machine base learners, the choice of base learner is not crucial and in fact any other single-instance learning algorithm could be used instead. For this reason, we classify DD-SVM and MILES as wrapper algorithms, and discuss them in more detail in Section 3.3.

3.2.4 Logistic Regression and Boosting

[Xu and Frank, 2004] presented multi-instance upgrades for the logistic regression and boosting algorithms, based on the collective assumption (described in Section 2.2.10), where each instance contributes equally and independently to the class label of its bag.

Logistic regression is a statistical method often employed for single-instance learning, where a linear model is built on a transformed version of the target variable. The transformation used is the *logit transformation*, which converts a probability into the logarithm of the odds of that probability.

In the single-instance case, the algorithm computes the instance-level class probabilities, which are not so easy to determine in multiple instance learning due to class labels only being provided at the bag level. However, the bag-level class probability function is easily computed under the collective assumption, and the instance-level probability function can be estimated from this by maximizing the binomial log-likelihood.

Xu and Frank used two variants of the collective assumption: in one version, the class label of a bag is determined by the geometric mean, and in the the other version the class label is determined by the arithmetic mean. In the case of the geometric mean, the algorithm effectively just converts the problem into a single-instance logistic regression problem, where each bag is represented by the mean of the instances in the bag. Unfortunately, this simplification does not apply when the arithmetic mean is used.

Boosting [Freund and Schapire, 1996] is a meta-learning algorithm that was developed based on work from computational learning theory. However, later analysis from a statistical perspective [Friedman et al., 1998] explained the algorithm as ad-

ditive logistic regression. Effectively, the algorithm estimates the log-odds function (as in logistic regression) based on an additive model. Given the relationship between the algorithms, Xu and Frank were able to upgrade the Adaboost.MI boosting algorithm using roughly the same method as for logistic regression.

[Zhou and Zhang, 2006] later formulated a method called MIMLBoost, which applies Xu and Frank's MI boosting method to multi-instance multi-label problems. MIMLBoost transforms the data into multi-instance multiclass data (where multiple class categories are allowed, but the categories are mutually exclusive) and directly applies Xu and Frank's method to the transformed dataset.

Note that these upgraded boosting algorithms can also be considered to be wrapper algorithms, as any single-instance learning algorithm capable of handling instance weights can be used as the base learner.

Another method for applying boosting algorithms to MI learning was investigated by [Auer and Ortner, 2004]. They presented a weak multi-instance algorithm to be used as a base learner for the standard Adaboost algorithm. Unlike Frank and Xu's approach, which uses single-instance base learners for the boosting algorithm, Auer and Ortner's method uses a multi-instance base classifier. The boosting algorithm is otherwise unchanged.

Auer and Ortner's weak MI learner finds a ball (i.e. hypersphere) that is designed to separate the positive bags from the negative bags. If any instance of a bag is within the ball, the bag is labeled as positive, otherwise it is labeled as negative. The method is designed to find the optimal ball with respect to classification accuracy on the training data. If the infinity norm $\|.\|_\infty$ is used for the distance measure, the balls become hypercubes. Auer and Ortner presented a variation of the algorithm using the infinity norm, which grows some dimensions of the hypercube into a larger hyper-rectangle. As before, the optimal hyper-rectangle is computed with respect to classification accuracy on the training data.

3.3 MI learning using Wrappers for Single-Instance Algorithms

Another approach to MI learning is to build general algorithms which are capable of applying any arbitrary single-instance learner to MI data. We call these methods "*wrapper*" algorithms, as they wrap around a given single-instance learning algorithm to create a new MI algorithm. Unlike the methods discussed in the previous section, the single-instance learner is not modified in any way. Instead, some process (known as a *propositionalization* method) is used to create a version of the data to which the supervised learning algorithm can be applied. The output of the single-instance algorithm is used in some way to generate bag-level predictions.

Multi-instance wrapper algorithms can be applied to both instance-based and metadata-based MI concepts (see Section 2.2.6). For instance-based MI concepts, the assumption is that instances are labeled via some process, and these labels determine the bag-level class labels. Wrapper algorithms that learn instance-based concepts typically apply single-instance learning directly to the instances inside the training bags. At prediction time, bag-level labels are assigned based on whatever MI assumption is used, such as the standard MI assumption.

In contrast, bag-level class labels for metadata-based MI concepts are assumed to be determined by a process in some single-instance feature space consisting of metadata extracted from the bags. Wrapper algorithms that learn this type of concept typically map each bag into the metadata-based feature space via some transformation, and then learn a single-instance model on the transformed feature space. Each instance in the metadata-based feature space corresponds to a bag. The single-instance base learner can then directly provide bag-level class labels.

The new research presented in this thesis is mainly concerned with wrapper approaches to MI learning. Therefore, existing wrapper approaches are discussed here in greater algorithmic detail than the other types of MI algorithms.

3.3.1 Using Summary Statistics for Propositionalization

Multi-instance learning problems can be easily converted into single-instance problems by replacing each bag with a feature vector consisting of summary statis-

tics derived from the instances in that bag. This method originates from a similar approach to propositionalization for relational data known as RELAGGS [Krogel and Wrobel, 2002]. We will follow [Dong, 2006], and refer to the summary statistics approach as *Simple MI*. An implementation of the version of the approach described by Dong resides in the WEKA data mining software suite [Witten and Frank, 2005].

Simple MI is, as the name suggests, a very simple method for applying single-instance learners to MI problems. This algorithm learns simple metadata-type concepts. First, each bag is mapped into a single-instance feature space consisting of summary statistics for that bag. A propositional learner is then applied to the instances in the transformed feature space. At classification time, new bags are mapped into the metadata feature space, and predictions are made by merely outputting the prediction of the single-instance learner for the transformed version of the bag.

Dong described three versions of Simple MI, each of which differs only in the type of summary statistics used for the single-instance feature space. The first two methods merely average the values of the instances in a bag for each dimension, using either the arithmetic and the geometric mean. Formally, the two methods can be defined as follows: if b is a bag with instances from feature space $\chi = (x_1, x_2, \ldots, x_n)$, then b is mapped to $(\bar{x_1}, \bar{x_2}, \ldots, \bar{x_n})$, where $\bar{x}$ is the arithmetic (or geometric) mean of the instances in the bag.

The third option is called the "minimax" method. Here, the minimum and maximum values of each variable are recorded for each bag. This method is equivalent to Gärtner et al.'s minimax kernel [Gärtner et al., 2002]. Using the same notation as before, each bag b is mapped to $(\min x_1, \min x_2, \ldots, \min x_n, \max x_1, \max x_2, \ldots, \max x_n)$. The new feature space contains $2n$ dimensions.

The main advantage of Simple MI is that it is extremely fast. The computation of the feature-space transformations are almost trivial, and the single-instance base learner only has to learn from as many instances as there were bags in the training set, regardless of how many instances were contained inside the bags. Of course, this simple model is not able to represent some types of problems. However, Dong found that Simple MI (with appropriate base learners) performs surprisingly well

on many datasets, even outperforming all of the special-purpose MI algorithms that were investigated in some cases.

3.3.2 MIWrapper

MIWrapper [Frank and Xu, 2003] is another simple wrapper approach that is very competitive with other MI algorithms on many benchmark datasets, including the *musk* problem. The algorithm uses the collective assumption [Xu, 2003], where each instance in a bag contributes equally and independently to the label of the bag.

The first step for MIWrapper is to collect all of the instances from all of the bags, and label each of them with the label of the bag that they came from. This effectively creates a propositional (i.e. single-instance) dataset. The algorithm then weights all of the instances so that each bag has equal total weight. A single-instance learner is applied to this propositional dataset. At classification time, the single-instance learner predicts class probabilities for all of the instances in the bag. The output is merely the average (arithmetic or geometric) of the predicted instance-level class probabilities.

The weighting method used by MIWrapper is designed to treat all examples in the training data equally. Therefore, each bag is given the same total weight, regardless of the number of instances in the bag. The simplest way to implement this would be to set all instances from each bag i (containing n_i instances) to be $\frac{1}{n_i}$. However, this can create large numbers of instances with very small instance weights, which some algorithms (such as the C4.5 decision tree learner) do not handle very well. Frank and Xu solve this problem by multiplying the weights of each of the instances by a constant factor, so that the total weight of all the instances is equal to the number of instances in all of the training bags. The resulting weight assignment for instance j from the ith bag is:

$$w_{ij} = \frac{m}{N} \times \frac{1}{n_i} \ , \tag{3.1}$$

where m is the total number of instances, and N is the number of bags.

Using the arithmetic mean at prediction time, the output is exactly the "sample" version of the collective assumption formula:

$$Pr(c|b) = \frac{1}{n_b} \sum_{i=1}^{n_b} Pr(c|x_i) \ ,$$

where c is a class value, b is a bag, n_b is the number of instances in bag b, and x_i is an instance in bag b.

The simple heuristic for finding the instance-level labels is unlikely to be correct in practice, and therefore is likely to introduce bias. However, Frank and Xu empirically showed that regardless of this, MIWrapper is often able to make accurate predictions, as it manages to find a good *decision boundary*. Other methods for estimating instance-level class labels would be a possible avenue for future research.

3.3.3 Mi-NB

The mi-NB algorithm [Murray et al., 2005] was designed for the hard-drive failure prediction problem (see Section 2.2.2), where an extremely low false-positive rate is required. Hard drives that are predicted as positive (i.e. failed or about to fail) generally must be replaced by the manufacturer under warranty conditions, so a large number of false positives would be very expensive. Therefore, the algorithm is designed to maintain a very low false-positive rate. The acronym "mi-NB" stands for multiple-instance naive Bayes. However, any other single instance learner that outputs class probabilities could be used as the base learner for the algorithm.

Murray et al. defined the initial state of the algorithm in terms of the hard-drive failure prediction problem. In their representation, hard-drives are represented by bags of drive status information feature vectors (SMART records) from time-series data recorded automatically by the hard-drives. They set the class labels of the most recent instances from failed drives (i.e. the status of the drives immediately before they failed) to be positive, and all other instances to be negative. Thus, very few instances will be positive, resulting in a low initial false-alarm rate. For other problem domains, some other heuristic would be required to initialize class values.

After the initial instance-level class values are set, an iterative procedure is applied. First, a naive Bayes model is built on all of the instances. The instances are then relabeled according to the predictions of this model. However, after relabeling, for each positive training bag that is misclassified (according to the standard MI assumption that a bag is positive if and only if it contains a positive instance), the

instance with the highest probability estimate for the positive class is relabeled as positive. The procedure is repeated until some stopping criterion is met, such as the false-positive rate exceeding a maximum value, or a certain number of iterations having been performed.

At classification time, all instances in a test bag are labeled by the naive Bayesian classifier. The bag-level classification is made using the standard MI assumption: bags are labeled as positive if and only if they contain a positive instance.

The naive Bayesian classifier used by mi-NB is an extremely simple statistical model. Naive Bayes often provides surprisingly high classification performance, despite the fact that a key assumption for the model (namely that the features are conditionally independent of each other given the class) is very rarely true in practice.

It should be noted that mi-NB could actually use a different classifier instead of naive Bayes for the base learner, and thus we classify it as a wrapper approach. However, because the base learner must be rebuilt after every iteration, the authors recommend that naive Bayes be used for computational efficiency reasons, at least for the hard-drive problem. Unlike most classifiers, naive Bayes can be efficiently updated when some instance class labels have been modified. Other base learners would need to be recomputed from scratch.

It appears that Murray et al. were concerned that the algorithm be very efficient as it may have to run on a hard-drive's very modest built-in processor. However, we believe that training time should not be an issue, because the algorithm need not be trained online on the hard-drive. Only the classification step need be performed on the hard-drive itself, and then for only a single bag (itself). Therefore, algorithms that are slow at training time but very efficient at classification time, such as support vector machines, could be used instead for the hard-drive problem. It is well known that naive Bayes produces poor class probability estimates when the conditional independence assumption does not hold, so the use of other base learners would be a promising area for research. Recent work on model compression [Buciluǎ et al., 2006], where a machine learning model is simplified for use on devices with limited processing power or resources, could also be applied in this situation. See Algorithm 1 for a pseudocode representation of mi-NB, reinterpreted as a general-purpose multi-instance wrapper algorithm.

Algorithm 1 mi-NB

D = the set of training bags
C = all instances in the bags in D
L = a single-instance base learner
$FA_{desired}$ = the desired false-alarm rate

$\underline{train(D)}$
Initialize class labels for C in some domain-specific way that provides a very low false-alarm rate
$L.train(C)$
for all $C_i \in C$ **do**
    $C_i.setClassValue(L.classify(C_i))$ //Classify instance C_i using L
FA = bag-level false alarm rate on training data
while $FA < FA_{desired}$ **do**
    // For all misclassified positive bags
    for all $D_i.getClass() = 1, L.classify(D_{ij}) = 0 \; \forall D_{ij} \in D_i$ **do**
        $j^* = \arg\max_j L.probabilityOfPositiveClass(D_{ij})$
        $D_{ij^*}.setClassValue(1)$ //Reclassify instance as positive
    $L.train(C)$
    for all $C_i \in C$ **do**
        $C_i.setClassValue(L.classify(C_i))$ //Classify instance C_i using L
    FA = bag-level false alarm rate on training data

$\underline{classify(B)}$, $B = \{x_i : i = 1, \cdots, |B|\}$ a test bag
// Classifies B according to the standard MI assumption
for all $x_i \in B$ **do**
    if $L.classify(x_i) = 1$ **then**
        return 1
return 0

3.3.4 Two-Level Classification

The Two-Level Classification (TLC) algorithm [Weidmann et al., 2003] is designed to learn the type of MI concepts that are described in Weidmann's concept hierarchy (see Section 2.2.7 for details).

These MI concepts consist of a set of instance-level concepts that are related in some way to the bag-level concepts. Following the tradition in the MI literature, the bag-level learning problem is a binary classification problem. However, there may be an arbitrary number instance-level concepts. It is assumed that bag-level class labels are determined by the *counts* of each instance-level concept in a bag.

TLC learns in a two-step process. The first step learns instance-level concepts using a decision tree. The tree is built on all of the instances in all of the bags in the training data, with class labels set to the labels of the parent bags. Instances are weighted in the same way as MIWrapper (using Equation 3.1), so that each bag to has the same weight and the total of the weights sums to the number of instances. Information gain is used for test selection. A simple prepruning heuristic is applied, where nodes are not split further when the sum of the instance weights in that node is less than two. No other pruning methods are applied.

Each node in the tree is considered to represent a concept. Then each bag is converted into a single-instance representation, with an attribute for every node in the tree (i.e. each concept), the value of which is set to the number of instances that reach that node in the decision tree.

The second step learns bag-level concepts, based on the instance-level concepts discovered in the first step. A single-instance learning algorithm is applied to the transformed data. The same mapping is performed at classification time, and the bag-level predictions are made by the single-instance learner.

A further (optional) refinement to the algorithm is to use attribute selection to try to eliminate the attributes that do not contribute to the instance-level classification problem learned by the decision tree. Weidmann et al. used the attribute selection procedure from [Kohavi and John, 1997], which evaluates a subset of features using cross-validation. They used *backward selection*, starting with all attributes and eliminating ones that worsen the performance of the algorithm. The evaluation is performed at the bag-level using both levels of the TLC algorithm. Obviously,

this process is very computationally expensive.

Weidmann et al. found that the TLC method (using boosted decision stumps as the base learner) was very competitive on the *musk* data, and also performed extremely well on artificial datasets where generative models from Weidmann's concept hierarchy were used.

3.3.5 MILES

Multiple-Instance Learning via Embedded Instance Selection (MILES) [Chen et al., 2006] is an approach to MI learning based on the diverse density framework. MILES embeds bags into a single-instance feature space, and applies the 1-norm support vector machine algorithm to the transformed dataset. However, other single-instance learning algorithms could be used in place of the 1-norm SVM, so we classify the algorithm as a wrapper approach.

Most earlier diverse density-based algorithms have used the standard MI assumption and further assume the existence of a single target point. MILES relaxes these assumptions. Instead, a symmetric assumption is used, where multiple target points are allowed, which may be related to either positive or negative bags. Under this assumption, and using the *most-likely-cause* estimator from the diverse density framework, Chen et al. define a measure that estimates the probability that a point is a target point given a bag, regardless of the bag's class label:

$$Pr(x|B_i) \;\propto\; s(x, B_i) = \max_j \exp\left(- \frac{\|x_{ij} - x\|^2}{\sigma^2} \right) , \tag{3.2}$$

where x_{ij} are the instances in bag B_i, and σ is a predefined scaling factor. Note that $s(x, B_i)$ can be interpreted as a measure of similarity between a bag and an instance, which is determined by the instance and the closest instance in the bag.

At this stage, the question remains as to how to find the target points. For the sake of computational feasibility, MILES uses the assumption that the target points can be approximated by instances in the training bags. In other words, each instance is a candidate for a target point. The candidates are represented as features in an instance based feature space $\mathbb{F}_c$. Each bag in the training set is mapped into $\mathbb{F}_c$ via the mapping

Algorithm 2 MILES

D = the set of training bags
C = all instances in the bags in D
L = a single-instance base learner
σ = the scaling factor, a parameter to the algorithm

$\underline{MILES\_transform(B)}$
for (every instance x^k in C) **do**
    $d = \min_j \|x_j - x^k\|$
    the kth element of $m(B)$ is $s(x^k, B) = e^{\frac{-d^2}{\sigma^2}}$
return $m(B)$

$\underline{train(D)}$
F = an empty set of instances
for (every bag $B_i = \{x_{ij} : j = 1, \cdots, n_i\}$ in D) **do**
    $t = MILES\_transform(B_i)$
    $t.setClassLabel(B_i.getClassLabel())$
    $F = F \cup \{t\}$
$L.train(F)$ // Can optionally perform feature selection here also

$\underline{classify(B)}$, $B = \{x_j : j = 1, \cdots, n_i\}$ a test bag
$t = MILES\_transform(B)$
return $L.classify(t)$

$$m(B_i) = [s(x^1, B_i), s(x^2, B_i), \cdots, s(x^n, B_i)]^T \ , \tag{3.3}$$

where $x^i \in C$ is an instance from the set C of all instances in all of the training bags. If the class labels $c \in \Omega$ of the bags are appended, the space $(\mathbb{F}_c|\Omega)$ is a single-instance feature space to which standard supervised machine learning algorithms can be applied. The output of this classifier can be used to provide bag-level class labels for future data. The pseudocode of the MILES algorithm is provided in Algorithm 2.

Chen et al. used the 1-norm SVM algorithm as the base classifier, due to the high dimensionality of the feature space. The 1-norm SVM is reasonably efficient computationally at learning from high-dimensional datasets because it can be trained using linear programming, and is known to generally set most feature weights to zero, which effectively performs feature selection. This saves on performing an additional feature-selection process, which may be computationally expensive.

1-Norm Support Vector Machines

Support vector machines (SVMs) were introduced earlier in the thesis, in Section 3.2.3. The reader should refer to that section for a brief introduction to support vector machine algorithms. The 1-norm SVM (see, for example, [Zhu et al., 2003]) is a variant on the standard support vector machine that uses an alternative metric for computing the ridge penalty. We will discuss here in more detail the 1-norm SVM formulation used by Chen et al. for MILES.

Recall that support vector machines construct a hyperplane over the instance space. The hyperplane is used as a decision boundary, which divides the space into a positive and a negative halfspace. This can be represented mathematically as

$$y = \text{sign}(w^T m + b) \ ,$$

where $y \in \Omega = \{+, -\}$ is the classification label output by the SVM, m is an instance in the feature space $\mathbb{F}_c$, and w and b are model parameters. Note that $w^T m + b$ is a linear equation. This equation determines the location of the hyperplane that separates the instance space. The sign function determines which side of the hyperplane m is on, and labels the output accordingly.

Support vector machine algorithms are designed to find the maximum margin hyperplane, which is placed at equal distance from both the positive and negative sets of instances, thus attaining the largest margin between it and the training points. This can be achieved by finding w^* and b^*, the optimum values of w and b with respect to minimizing the regularized training error:

$$\lambda P[.] + \varepsilon_{\text{training}} \ ,$$

where $P[.]$ is a regularizer, λ is the regularization parameter, and $\varepsilon_{\text{training}}$ is the training error. The training error is often defined as the total of the losses that each instance m introduces via a hinge loss function:

$$\varepsilon = \max\{1 - y(w^T m + b), 0\} \ .$$

The regularizer is used to prevent overfitting by penalizing overly complex models. In standard SVM formulations, the regularizer is the squared 2-norm (Euclidean

norm) of the weight vector, $\|w\|$. Under this formulation, the problem of finding the maximum-margin classifier can be formulated as a quadratic programming (QP) problem. This problem can be solved using standard QP optimization packages, or special-purpose QP solving algorithms that are optimized for support vector machines, such as Sequential Minimal Optimization [Platt, 1998].

However, Chen et al. use the 1-norm SVM formulation for MILES. Here, the regularizer is the 1-norm (Manhattan length) of the weight vector. The 1-norm penalty causes more weights to be set to zero than the 2-norm, and thus the 1-norm SVM is also known as the sparse SVM. The 1-norm penalty can be written as:

$$\|w\|_1 = \sum_k |w_k| \ .$$

Using the 1-norm of w as the regularizer, the optimization problem becomes a linear programming (LP) problem. LP problems involve optimizing a linear equation under a set of linear constraints. They can generally be solved more quickly than quadratic programming problems.

There is an additional minor restriction on LP problems in that all of their variables must be greater than zero. However this condition is easily met by replacing any potentially negative variables with a pair of variables, where one is subtracted from the other. In the case of the 1-norm SVM, the weight vector w may contain negative values, and so is replaced by $w = u - v$, where $u_k, v_k \geq 0$. Let ε and η be the hinge losses for the l^+ positive instances and the l^- negative instances, respectively, and let μ be a parameter that can be used to bias towards the minor class if the data is skewed. Recall that λ is the regularization parameter, which is an input parameter to the model. Then the 1-norm SVM can be formulated as the following linear program:

$$
\begin{aligned}
\min_{u,v,b,\varepsilon,\eta} \quad & \lambda \sum_{k=1}^{n}(u_k + v_k) + \mu \sum_{i=1}^{l^+} \varepsilon_i + (1-\mu) \sum_{j=1}^{l^-} \eta_j \\
\text{s.t.} \quad & [(u-v)^T m_i^+ + b] + \varepsilon_i \geq 1, i = 1, \cdots, l^+, \\
& -[(u-v)^T m_j^- + b] + \eta_j \geq 1, j = 1, \cdots, l^-, \\
& u_k, v_k \geq 0, k = 1, \cdots, n, \\
& \varepsilon_i, \eta_j \geq 0, i = 1, \cdots, l^+, j = 1, \cdots, l^- \ .
\end{aligned}
$$

Interpretation of MILES as Learning Instance Weights

Chen et al. describe MILES from a feature selection point of view. However, when features correspond to instances, feature selection is effectively the same as instance selection. In this section, we show how MILES can be interpreted as learning instance weights, and thus is closely related to the new models and algorithms presented in this thesis.

MILES builds a 1-norm SVM over the instance-based feature space $(\mathbb{F}_c|\Omega)$. The 1-norm SVM learns the weight vector w^*, which assigns a weight value to each feature, the absolute value of which determines the level of influence that the feature has on the classification label.

In $\mathbb{F}_c$, each feature represents an instance from a training bag. Since each feature is given a weight by the 1-norm SVM, each instance in the training data is effectively given a weight. From this point of view, it is easy to see that MILES ascribes weights to each instance in the training data.

Although MILES finds weights for the training instances, and uses this to determine the level of influence of different parts of instance space, it does not create a well-defined weight function over instance space. The similarity function $s(x, B)$ between a training instance and a bag is influenced only by the instance in B that is the closest to the training instance x. If an instance in B is never the closest instance in the bag to a training instance, it will have no effect on the bag's classification label.

In order to better understand what is going on here, let us temporarily circumvent the problem by considering a bag B that contains only a single instance, $x \in \chi$.

Consider an instance x^k from the training data. The instance x^k influences the weight of x via a radial influence function $(w^*[k])e^{\frac{-d^2}{\sigma^2}}$, where $w^*[k]$ is the weight assigned to x_k by the SVM, d is the distance from x^k to x and σ is a predetermined scaling constant. The classification y of the bag B is determined by the SVM's linear classification function:

$$\begin{aligned} y &= \text{sign}(w^{*T}.m(x) + b^*) \\ &= \text{sign}((\sum_k w^*[k].e^{\frac{-\|x^k - x\|^2}{\sigma^2}}) + b^*) \\ &= \text{sign}(w_f(x) + b^*), w_f(x) = (\sum_k w^*[k].e^{\frac{-\|x^k - x\|^2}{\sigma^2}}) \ . \end{aligned}$$

Here, $w_f(x)$ is the influence that the instance x has on the class label of the bag, which is otherwise determined by the constant b^*. In this sense, it is an instance weight. Because w^*, b^*, e, σ and the training instances $x^k \in C = \{x^k : k = 1, \cdots, n\}$ are all constants, w_f is clearly a function over the instance space.

However, when extra instances are added to the bag, the contribution of each instance to the class label is no longer dependent only on the feature values of that instance. Recall that the computation of $m(B)$ (using Equation 3.3) involves the similarity function $s(x^k, B)$. The similarity function (Equation 3.2) includes a $\max_j$ term over all of the instances in the bag, which produces a value related only to the instance in B that is closest to the candidate target point x^k.

Because of this maximization step, the contribution that an instance has towards a bag's class label depends on the other instances in the bag as well. An instance that is not the closest in the bag to x^k will not contribute at all to the value for feature k. In this case, w_f is not a function over χ, as it depends on B as well. Thus it can be seen that for MILES, the level of influence an instance has on the bag's class label depends not only on the weights learned at training time, but also on the other instances in the bag.

MILES learns instance weights for training instances, and correspondingly learns a radial influence function for each training point. In this sense, it can be said to learn instance weights. But even though each instance has its own influence function over instance space, these functions are combined in a bag-dependent way, resulting

in a bag-dependent weight function $w_f(x) : \chi \times \mathbb{B} \rightarrow \mathbb{R}$ from the Cartesian product of instance space χ and bag space $\mathbb{B}$ to the real numbers.

In contrast, in this thesis we formulate a type of MI concept where the weight of a point is determined only by the attribute values of that point, and is not dependent on the rest of the bag. In other words, $w_f(x) : \chi \rightarrow \mathbb{R}$ is a function over instance space. Chapter 5 introduces a new algorithm based on MILES that learns such a weight function.

3.3.6 DD-SVM

The diverse density support vector machine (DD-SVM) algorithm [Chen and Wang, 2004] is a predecessor to MILES that was designed by the same authors. Chen et al. (2006)'s experimental results show that MILES is much more efficient than DD-SVM in terms of computational complexity, while maintaining similar or better classification accuracy and increased robustness to label noise.

Despite a very high degree of similarity between the approaches, the connection between DD-SVM and MILES is not mentioned in [Chen et al., 2006]. Here, we fill this gap in the literature by describing the similarities between MILES and DD-SVM, and detailing the points of difference.

Like MILES, DD-SVM performs a feature space mapping, where the distance from a target point to the closest instance in a bag is represented as an attribute in a single-instance feature space. Each bag is mapped into this feature space, effectively converting the problem into a single-instance one. A standard support vector machine is then applied (as opposed to the 1-norm SVM used by MILES). However, any single-instance learning algorithm could have been applied instead of the SVM, so we view the algorithm as a wrapper approach.

The key difference to MILES is the selection of instance prototypes for use as attributes in the feature space mapping. MILES uses every instance in the training bag as an instance prototype. However, in DD-SVM, instance prototypes are local maxima in the diverse density function. Similarly to the maxDD algorithm, each positive instance is used as a starting point for an optimization procedure (in this case a quasi-Newtonian search) that finds a local maximum, which is selected as a

candidate instance prototype. As DD-SVM is designed to disregard the standard MI assumption, an optional extra step is to reverse the class labels and repeat this process. The candidate instance prototypes are filtered for distinctness, and those with diverse density below a given threshold are discarded.

Another small difference is the similarity function between bags and instance prototypes. In DD-SVM, the similarity function is $\min_{j=1,\ldots,N_i} \|x_{ij} - x\|w$, where x_{ij} are the instances in the bag, x is the instance prototype and w is a weight vector determining the importance of each feature. Here, the absolute value of the distance is used directly in the computation, whereas a Gaussian function is used in MILES.

Chen and Wang use a standard 2-norm SVM for DD-SVM's single-instance base learner, as opposed to the 1-norm SVM used in MILES. Because the instance prototypes are filtered for distinctness and size of diverse density, the feature space is generally smaller than the feature space for MILES. For this reason, the sparseness property of the 1-norm was presumably not considered necessary.

The main computational advantage that MILES has over DD-SVM is the avoidance of the expensive optimization procedure over the diverse density function, which must be performed for every single instance in DD-SVM. Chen et al's experiments with MILES show that this optimization step is unnecessary.

This is a very interesting result. If candidate target concepts can be represented directly by the instances in the training bags instead of diverse density maxima without loss of classification performance, this modification could prove to be extremely effective for the maxDD algorithm also. By restricting the search for diverse density maxima to instances from positive bags, the expensive gradient ascent optimization procedure could be avoided. Given MILES' good performance without the optimization step, we conjecture that the maxDD algorithm will work well using this heuristic, despite the vastly reduced computational complexity. We intend to investigate this in the future.

As well as the computational improvement with respect to DD-SVM, Chen et al. additionally showed empirically that MILES is more robust with respect to label noise than the earlier method. Because of these superiorities enjoyed by MILES compared to DD-SVM, we will not consider DD-SVM any further.

Chapter 4

MILES as a Meta-Classifier

This chapter presents an empirical study of the performance of the MILES algorithm using a variety of single-instance base learners on a diverse set of real-world datasets. For comparison, we also provide experimental results for other MI algorithms. The goal of the study is to compare the relative performance of different base learners for MILES, and to compare MILES to existing MI algorithms.

Although Chen et al. (2006) provide some empirical results of the algorithm on the *musk* datasets and for an image categorization problem, these results all use the 1-norm SVM as the base learner. In fact, Chen et al. actually include the 1-norm SVM as part of the definition of MILES, although they do mention briefly that other base learners are possible.

In this chapter we view the algorithm as a meta-classifier that can wrap around an arbitrary single-instance learner. When other base learners are used instead of the 1-norm SVM, instance weights may not be learned explicitly. However, due to the instance-based feature space representation used by MILES, alternative base learners will still implicitly determine which areas of instance space are important, as they must infer some connection between attributes (which correspond to points in instance space) and class labels. Hence, these learners should be able to solve similar types of learning problems, and thus are of interest for this thesis.

4.1 Experiment Design

An extensive set of experiments was performed on a number of multi-instance datasets, using a wide range of MI algorithms and single-instance base learners. The experiments were performed using the WEKA workbench [Witten and Frank, 2005].

Each algorithm was evaluated on each dataset via repeated 10-fold cross-validation (CV). n-fold cross-validation is a standard statistical evaluation technique

where the dataset is divided into n subsets of equal size, called *folds*. The algorithm is repeatedly trained on the instances in $n-1$ of the folds, with the remaining fold used as the test set. The results are averaged over the n executions. We performed ten repeats of the 10-fold CV, and averaged the results over all of the repeats.

Performance was measured using classification accuracy. We tested for significant differences between algorithms using the corrected resampled t-test [Nadeau and Bengio, 2003] with significance level $\alpha = 0.05$.

4.1.1 Algorithms

The implementations provided in WEKA were used for all of the MI algorithms and single-instance base learners, with the exception of MILES and the 1-norm SVM which were implemented specifically for the experiment[1]. The default parameters in WEKA were used for each algorithm unless otherwise specified. A list of the algorithms follows, with very brief descriptions provided.

- *MISMO* Support vector machine algorithm using the MI polynomial kernel [Gärtner et al., 2002], trained with the Sequential Minimal Optimization [Platt, 1998] method (see also Section 3.2.3).

- *mi-SVM* The Maximum Pattern Margin support vector machine formulation [Andrews et al., 2002] (see also Section 3.2.3).

- *Citation KNN* Citation K-Nearest Neighbours [Wang and Zucker, 2000], using one citer and one reference, and the rank-1 Hausdorff Distance (see also Section 3.2.1).

- *EMDD* The Expectation-Maximization Diverse Density [Zhang and Goldman, 2002] algorithm (see also Section 3.1.2).

- *Adaboost + Opt. Ball* Ten iterations of Adaboost.M1 [Freund and Schapire, 1996] with the Optimal Ball [Auer and Ortner, 2004] weak MI base learner (see also Section 3.2.4).

[1]The implementations of MILES and the 1-norm SVM will be included in future versions of WEKA.

- *MIBoost* Adaboost.M1 upgraded for MI learning [Xu and Frank, 2004]. A simple information-gain based decision tree learner implemented as *REPTree* in WEKA was used as the base learner. Automatic pruning was turned off, and trees were instead restricted to a depth of 3 levels. Ten boosting iterations performed (see also Section 3.2.4).

- *MILR* Logistic regression upgraded for MI learning [Xu and Frank, 2004], using the standard MI assumption based on the noisy-or model [Maron, 1998] (see also Section 3.2.4).

- *MIWrapper* The MIWrapper algorithm [Frank and Xu, 2003] using the arithmetic average of the instance-level class probabilities for estimating bag-level class probabilities (see also Section 3.3.2).

- *SimpleMI* Propositionalization [Dong, 2006] by replacing each bag with the arithmetic average of its instances (see also Section 3.3.1).

- *MILES* Multiple-Instance Learning via Embedded Instance Selection [Chen et al., 2006], with $\sigma^2 = 8 \times 10^5$ as used by Chen et al. for the *musk2* dataset (see also Section 3.3.5).

The following single-instance base learners were used for MIWrapper, SimpleMI and MILES:

- *C4.5* The standard C4.5 [Quinlan, 1993] decision tree induction algorithm.

- *RandomForest* The Random Forest [Breiman, 2001] ensemble decision tree algorithm, with 100 trees.

- *Adaboost + C4.5* The Adaboost M1 ensemble method with C4.5 as the base learner, ten iterations.

- *Adaboost + D. Stump* The Adaboost M1 ensemble method with a decision stump (1-level decision tree) as the base learner, 100 iterations.

- *Bagging + C4.5* The Bagging [Breiman, 1996] ensemble method with an unpruned C4.5 decision tree as the base learner, 10 iterations.

- *SMO (LIN)* Support vector machine trained via sequential minimal optimization [Platt, 1998], using a linear kernel. Logistic models were fitted to the outputs to achieve better probability estimates.

- *SMO (RBF)* Support vector machine trained via sequential minimal optimization [Platt, 1998], using a radial basis function kernel. Logistic models were fitted to the outputs to achieve better probability estimates.

- *1-Norm SVM* The 1-norm support vector machine as formulated by [Chen et al., 2006], with $\lambda = 10^{-4}$, $\mu = 0.5$. The selection of the λ value is explained in Section 4.2.1. The linear programming optimization was solved by the *LPSolve*[2] software package.

- *Logistic* Multinomial logistic regression, using a ridge estimator [le Cessie and van Houwelingen, 1992].

4.1.2 Datasets

The datasets used in this experiment correspond to a large subset of the application domain problems described in Section 2.2.2. The reader is referred to that section for more detail on the problem domains and the multi-instance representations used. The datasets are described very briefly here, with the names of the datasets (as labeled in the results tables) italicized for convenience.

Drug Activity Prediction

The *musk1* and *musk2* datasets are the MUSK data used in [Dietterich et al., 1997]. Each bag represents a molecule, and the task is to predict whether the molecule emits a musky odour.

Inductive Logic Programming

Eastwest is the train direction prediction problem from the East West Challenge ILP contest [Michie et al., 1994]. Each bag represents a train, and the task is to predict whether the train is Eastbound or Westbound. *Westeast* is exactly the same problem as *eastwest*, except that the class labels are reversed. This is an interesting

[2]Publicly available for download from http://sourceforge.net/projects/lpsolve

variation because *eastwest* is compatible with the standard MI assumption, while *westeast* is not [Dong, 2006].

The mutagenicity prediction problem [Srinivasan et al., 1994] was also used in the experiments. Three representations proposed by [Reutemann, 2004] for transforming the mutagenesis ILP problem into a multi-instance problem were used. These representations were briefly described earlier in Section 2.2.2. The datasets are labeled *mutagenesis-atoms*, *mutagenesis-bonds* and *mutagenesis-chains*.

The *suramin* dataset [Braddock et al., 1994] is another ILP-based drug activity prediction problem, where the task is to detect suramin analogues that can act as anti-cancer agents.

Identifying Thioredoxin-fold Proteins

The *thioredoxin* dataset is the thioredoxin-fold protein identification task proposed by [Wang et al., 2004]. In this problem, the aim is to identify proteins belonging to the thioredoxin-fold protein superfamily.

Content-based Image Retrieval

Two sets of image data for CBIR tasks were used, each containing three different target image categories. These image databases provided six different image retrieval problems — one for each image category.

The first image database was originally provided by [Andrews et al., 2002], and contains MI bags representing photographs of *elephants*, *foxes* and *tigers*. The images were originally from the Corel dataset (published by COREL corporation), and were preprocessed and segmented using the Blobworld system [Carson et al., 1999]. Features from the segments were extracted, representing colour, texture and shape information. A problem dataset was created for each target animal, containing bags representing 100 images of the target animal and 100 randomly selected images of other animals.

The second CBIR dataset was the GRAZ02 [Opelt et al., 2006] dataset, containing images of *bikes*, *cars* and *people*. The target objects in this dataset are not always prominent in the image, may be occluded, and vary in scale. Furthermore, the images were carefully selected with respect to background, so that similar backgrounds

occur for all image categories. A dataset was created for each image category, containing target images and a similar number of *counter-class* images that contained no bikes, cars or people.

[Mayo, 2007] extracted 72 features derived from the Ohta colour space representation of the images. The features included the mean, median, mode, minimum, maximum, standard deviation, skewness and kurtosis values of each of the three components of the Ohta colour space representation, and a normalized histogram of the frequency of pixels within 16 intensity ranges for each of the Ohta space components. Although Mayo only describes a single-instance representation in the 2007 publication, he also created a multiple-instance representation of this dataset by segmenting the images into equal-sized blocks, and applying the above feature extraction process to each block. This is the data used for the experiments here.

4.2 Experimental Results and Analysis

This section presents the results of the experiments, and discusses their implications. The initial parameter tuning for MILES with the 1-norm SVM is described. A comparison of base learners for MILES is provided, and finally MILES is compared with the other wrapper algorithms, purpose-built MI algorithms and upgraded MI versions of single-instance algorithms.

4.2.1 Parameter Tuning for the 1-Norm SVM

Given the number of algorithms and datasets investigated, detailed parameter tuning for all of the MI algorithms and base classifiers was infeasible. With the exception of the 1-norm SVM, the classification schemes used in the experiment had fairly robust parameter values already provided by the default settings in their WEKA implementations. However, because the 1-norm SVM was implemented specifically for these experiments, a reasonable value for the regularization parameter λ was not known in advance.

Initially the value provided by [Chen et al., 2006] for the *musk2* dataset was used ($\lambda = 0.45$), with *musk2* being the larger of the two *musk* datasets. Unfortunately, MILES with the 1-norm SVM algorithm using this λ value produced poor results

on many datasets — in particular, the image classification datasets and the ILP *eastwest* / *westeast* problems. On these datasets, all of the attribute weights for the linear model learned by the SVM were set to zero. The result was typically a classifier that was only able to achieve 50 % accuracy according to the 10×10-fold cross-validation results, i.e. no better than chance.

In contrast to these observed results, [Chen et al., 2006] state that MILES is relatively stable with respect to the σ and λ parameters. However, they did not evaluate MILES on such a wide variety of datasets, which may be the reason for this discrepancy. Note also that in our implementation the freely available open-source *LPSolve* linear programming software package was used to solve the LP problem for the 1-norm SVM, while Chen et al. used the commercial package *CPLEX 9.0*. It is possible that a commercial LP solver such as *CPLEX* would not suffer from this particular issue, as the commercial software may have superior numeric stability over the open-source solution.

In any case, a new λ value was required. Because the experiments were performed using the cross-validation evaluation method, there were no separate training or validation datasets available to use for parameter tuning. Instead, parameter tuning for λ was performed on the *elephant* dataset. Six candidate values of λ were each tested via a single ten-fold cross-validation, with the first value being $\lambda_0 = 0.1$, using the step procedure $\lambda_{i+1} = \lambda_i/10$. It was found that $\lambda = 10^{-4}$ and $\lambda = 10^{-5}$ gave the minimum ten-fold cross-validation error rate on this dataset. The value $\lambda = 10^{-4}$ was selected because it corresponds to heavier regularization, and a smaller number of non-zero coefficients. The results of the parameter tuning experiment are shown in Figure 4.1. A comparison between the results for the original value of λ and the tuned value is provided in Table 4.1.

It should be noted that the results reported for the *elephant* dataset may be slightly optimistic, given that the test data for this dataset was used for parameter tuning. However, given that only a small number of values were tried, the bias should be fairly small. Another problem is that the decision to perform parameter tuning was made after seeing the results on all of the datasets. This is also, in a small way, "peeking" at the test data. These issues should be kept in mind when interpreting the results for MILES with the 1-norm SVM.

Figure 4.1: Parameter Tuning: λ Parameter Values vs Percentage Accuracy for the Elephant Dataset (10-Fold CV)

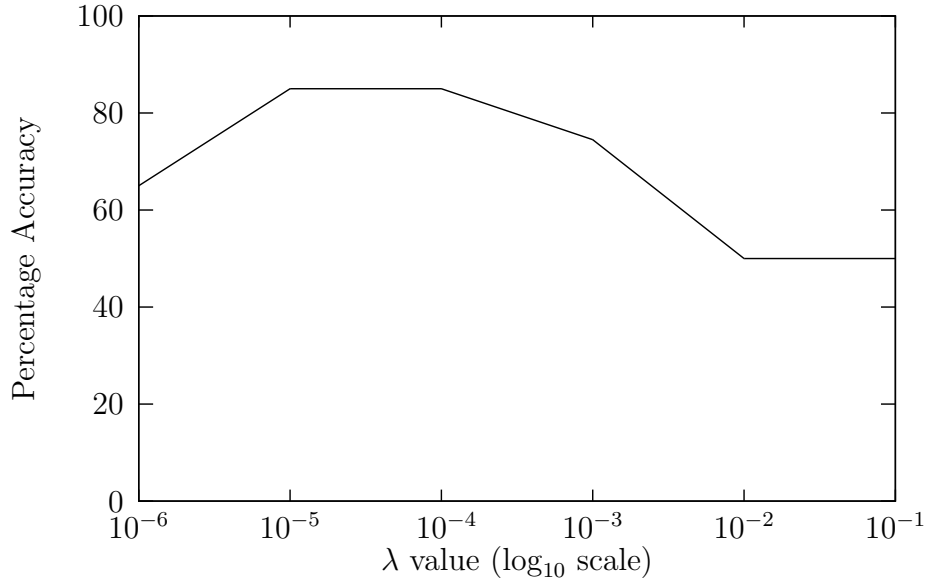


Table 4.1: Percentage Accuracy for MILES with 1-Norm SVM, Before and After Parameter Tuning

Dataset	Before tuning ($\lambda = 0.45$)	After tuning ($\lambda = 10^{-4}$)
musk1	85.1± 10.8	83.2±11.7
musk2	87.5± 9.7	90.4± 9.3
eastwest	50.0± 0.0	50.0± 0.0
westeast	50.0± 0.0	50.0± 0.0
mutagenesis-atoms	66.5± 2.3	77.4± 9.3 ∘
mutagenesis-bonds	66.5± 2.3	73.1±13.0
mutagenesis-chains	66.5± 2.3	77.0± 9.0 ∘
suramin	65.0± 45.2	65.0±45.2
thioredoxin	88.1± 5.1	88.1± 5.1
elephant	50.0± 0.0	84.3± 8.7 ∘
fox	50.0± 0.0	61.0± 9.5 ∘
tiger	50.0± 0.0	82.0± 8.5 ∘
bikes	74.0± 4.8	76.1± 5.0
cars	68.8± 4.3	66.5± 5.5
people	70.8± 4.5	71.1± 4.9

∘, • statistically significant improvement or degradation vs $\lambda = 0.45$

4.2.2 Comparison of Base Learners for MILES

A major goal of the experiment was to compare different base learners for MILES, particularly with respect to the 1-norm SVM. The results of this part of the experiment are displayed in Tables 4.2 and 4.3.

It was found that the 1-norm support vector machine was competitive against the other base learners on all datasets except *eastwest* and *westeast*, with no other non-ensemble base learner consistently performing significantly better than it. However, the 2-norm support vector machines were superior on the GRAZ02 problems, using both the linear and the radial basis function kernels. Also, Adaboost with decision

Table 4.2: MILES: Percentage Accuracy for Non-Ensemble Base Learners

Dataset	1-Norm SVM	C4.5	SMO (LIN)	SMO (RBF)	Logistic
musk1	83.2±11.7	84.1±11.9	86.6± 9.9	76.1±14.6	84.8±11.3
musk2	90.4± 9.3	82.5±12.1	88.6±10.1	76.3±11.7 •	85.8±11.0
eastwest	50.0± 0.0	50.0± 0.0	54.0±33.1	80.0±24.6 ○	64.5±29.6
westeast	50.0± 0.0	50.0± 0.0	54.5±32.6	80.0±24.6 ○	68.5±33.1
mutagenesis-atoms	77.4± 9.3	80.8± 8.1	81.5± 8.2	82.7± 8.8 ○	83.8± 7.2
mutagenesis-bonds	73.1±13.0	77.1± 9.8	81.3± 9.7	79.1± 8.8	80.2± 8.8
mutagenesis-chains	77.0± 9.0	79.3± 9.5	77.6± 8.2	76.6± 9.6	73.5± 9.4
suramin	65.0±45.2	65.0±45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	88.1± 5.1	84.3± 7.1	69.0±10.6 •	87.1± 2.4	87.1*± 3.9
elephant	84.3± 8.7	77.5± 9.2	83.9± 8.0	52.8± 5.7 •	79.6± 9.1
fox	61.0± 9.5	56.8±11.2	61.8± 9.4	54.8± 5.6	63.6± 8.9
tiger	82.0± 8.5	69.7± 9.3 •	80.8± 8.8	63.8± 6.1 •	80.0± 9.2
bikes	76.1± 5.0	72.5± 5.7	80.5± 4.7 ○	77.2± 4.5	72.4± 4.8 •
cars	66.5± 5.5	62.6± 4.7	72.5± 4.9 ○	71.0± 4.6 ○	63.9± 4.9
people	71.1± 4.9	69.8± 5.8	74.9± 4.9 ○	75.3± 4.5 ○	66.9± 5.0 •

○, • statistically significant improvement or degradation vs 1-norm SVM
∗ We were unable to allocate enough memory to run MILES + Logistic on the thioredoxin dataset. This result was obtained using an alternative logistic regression algorithm, implemented as *SimpleLogistic* in WEKA. The algorithm uses LogitBoost to learn a logistic regression model. See [Landwehr et al., 2003] for more information.

Table 4.3: MILES: Percentage Accuracy for Ensemble Base Learners

Dataset	1-Norm SVM	Adaboost + D. Stump	Random Forest	Adaboost + C4.5	Bagging + C4.5
musk1	83.2±11.7	88.0±11.6	87.0±11.4	85.8±12.0	86.0±11.5
musk2	90.4± 9.3	83.2±11.5	81.7±11.2 •	83.2±11.3	83.7±11.5
eastwest	50.0± 0.0	81.0±24.4 ○	80.0±24.6 ○	50.0± 0.0	50.5± 5.0
westeast	50.0± 0.0	81.0±24.4 ○	80.0±24.6 ○	50.0± 0.0	50.5± 5.0
mutagenesis-atoms	77.4± 9.3	83.9± 8.6	82.0± 8.2	79.5± 8.5	80.5± 7.7
mutagenesis-bonds	73.1±13.0	86.3± 7.4 ○	79.7±10.5	80.1± 9.9	77.4± 8.9
mutagenesis-chains	77.0± 9.0	86.0± 8.0 ○	80.4± 9.2	80.8± 8.1	79.8± 9.1
suramin	65.0±45.2	65.0±45.2	65.0±45.2	65.0±45.2	62.0±46.1
thioredoxin	88.1± 5.1	89.3± 4.0	87.7± 2.7	85.6± 6.4	88.2± 4.6
elephant	84.3± 8.7	80.9± 7.7	82.3± 8.2	81.5± 8.9	84.0± 8.3
fox	61.0± 9.5	61.6±10.9	64.9±10.2	59.4±11.6	61.4±10.3
tiger	82.0± 8.5	80.5± 8.9	78.6± 9.0	75.4± 9.3	75.7± 8.4
bikes	76.1± 5.0	78.0± 5.0	79.2± 4.4	78.0± 4.5	77.7± 5.1
cars	66.5± 5.5	71.6± 4.1 ○	71.7± 4.0 ○	69.3± 5.0	70.5± 4.9
people	71.1± 4.9	75.6± 4.6 ○	77.5± 4.3 ○	75.4± 4.8 ○	76.6± 4.7 ○

○, • statistically significant improvement or degradation vs 1-norm SVM

stumps had no significant losses and several wins versus the 1-norm SVM.

The 2-norm SVM with the RBF kernel was the only support vector machine approach that performed well on the *eastwest* / *westeast* problems. It was not consistently strong, however, with sub-par performance on the *musk* and *corel* datasets. The 2-norm support vector machine with the linear kernel was very competitive with the 1-norm method, except for a poor result on the *thioredoxin* problem.

These results indicate that the 1-norm support vector machine may have no classification performance advantage over its 2-norm cousins as a base learner for MILES. In the observed experimental results, the sparsity property of the 1-norm SVM did not translate into consistently superior classification accuracy, despite the high dimensionality of the datasets produced by the MILES transformation.

The *eastwest* and *westeast* datasets were problematic for many MILES base learners, with half of the schemes performing little or no better than chance on these problems, although several schemes achieved accuracies of around 80%. Note that the results were similar or identical for both datasets, regardless of the base learner. This is as expected, given that MILES is designed to use a symmetric generalized MI assumption.

MILES' performance was consistent on the suramin problem. All base learning schemes achieved an accuracy of 65.0% on this dataset, except for Bagging with C4.5, where an accuracy value of 62.0% was observed. The small size of the dataset at least partially explains the consistency between schemes — it contains only 11 bags, albeit with many instances in each of those bags.

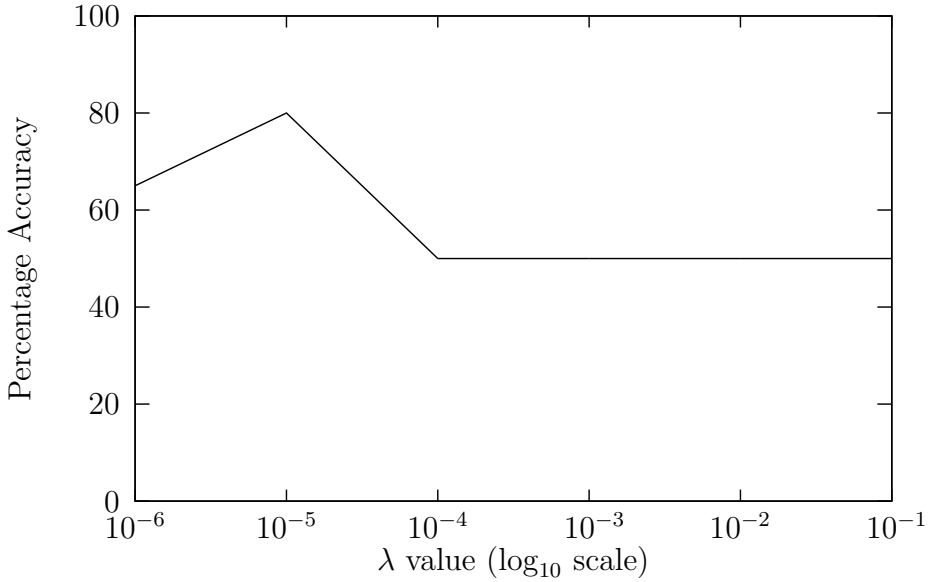
The ensemble methods were the strongest performers overall. Adaboost with decision stumps and random forests were the standout schemes in the MILES experiments, with six and four significant wins against the 1-norm SVM respectively. Although the 1-norm SVM was seven percentage points ahead of Adaboost for *musk2*, the difference was not statistically significant, while the random forests base learner incurred a significant loss against the 1-norm SVM on this dataset. Adaboost with decision stumps had higher percentage accuracy results than random forests on nine datasets and lower accuracy on five datasets, but none of those differences were statistically significant.

The strong performance of Adaboost.M1 with decision stumps is interesting, given the relationship between this model and the support vector machine model recommended by Chen et al (2006). Like support vector machines, the hypothesis learnt by Adaboost.M1 is a weighted linear threshold, i.e. a model of the form:

$$v(x) = sign\left(\sum_t \alpha_t h_t(x)\right) .$$

For Adaboost.M1, the h_ts are so-called *weak learners*. When decision stumps are used, each h_t corresponds to an attribute that the decision stump weak learner splits on. In this case, the choice of attributes for the decision stumps to split on is effectively attribute selection, similarly to the selection of attributes with non-zero weights made by the 1-norm support vector machine. The α_ts determine the importance of the corresponding decision stump, and therefore the importance of the

Figure 4.2: Parameter Tuning: λ Parameter Values vs Percentage Accuracy for the Eastwest Dataset (10-Fold CV)



attribute selected by that decision stump. Thus, the α_ts are effectively attribute weights, and perform a similar function to the attribute weights learned by the support vector machine model. Just as for the SVM, these attribute weights correspond to instance weights for the training instances, due to the instance-based nature of the feature space generated by the MILES transformation.

MILES with the 1-Norm SVM on Eastwest

Given that MILES with the 1-norm SVM as the base learner performed poorly on the *eastwest* / *westeast* problems, a separate parameter tuning experiment was used to investigate whether the result was due to a fundamental limitation in the algorithm, or merely an incorrect value for the λ parameter for these problems. The λ parameter was varied between 10^{-1} and 10^{-6} using the step procedure $\lambda_{i+1} = \lambda_i/10$, and the resulting classifiers were each evaluated using a single 10-fold cross-validation.

It was found that MILES with the 1-norm SVM achieved 80.0% accuracy when λ was set to 10^{-5} — a very similar result to the best-performing MILES base learners on this dataset. This percentage accuracy result of 80.0% was duplicated in a 10 $\times$ 10-fold cross-validation experiment on both *eastwest* and *westeast* datasets. Thus, it appears that MILES with the 1-norm SVM is capable of learning this type of problem when appropriate parameter values are used. The results of the parameter tuning experiment are displayed in Figure 4.2.

Table 4.4: MIWrapper: Percentage Accuracy for Non-Ensemble Base Learners

Dataset	1-Norm SVM	C4.5	SMO (LIN)	Logistic	SMO (RBF)
musk1	83.6±11.7	84.3±11.7	86.9±11.8	78.9±12.5	80.8±12.2
musk2	78.6±13.1	80.1±11.3	81.2±11.8	81.7±12.7	82.8±11.8
eastwest	47.0±28.3	52.0±35.5	53.0±33.2	61.5±33.2	68.0±29.7
westeast	47.5±29.6	49.5±32.2	53.0±33.2	61.5±33.2	68.0±29.7
mutagenesis-atoms	66.5± 2.3	76.4± 8.3 ○	66.5± 2.3	66.5± 2.3	66.5± 2.3
mutagenesis-bonds	66.5± 2.3	80.7± 8.9 ○	66.5± 2.3	67.2± 3.3	66.5± 2.3
mutagenesis-chains	66.7± 2.6	84.9± 7.2 ○	68.0± 3.8	70.6± 6.5	69.6± 3.5 ○
suramin	65.0±45.2	65.0±45.2	35.0±45.2	65.0±45.2	35.0±45.2
thioredoxin	87.1± 2.4	87.8± 2.7	87.1± 2.4	87.1± 2.4	87.1± 2.4
elephant	83.4± 8.0	80.3± 7.7	84.5± 8.7	84.0± 8.6	82.0± 8.4
fox	58.9± 9.6	64.1±10.6	59.2± 9.5	58.4±10.2	57.2±10.4
tiger	77.7± 8.1	77.4± 9.2	80.5± 8.0	78.4± 8.3	81.0± 8.0
bikes	81.9± 3.9	78.8± 4.6 ●	81.5± 4.1	82.2± 3.8	76.8± 4.3 ●
cars	72.3± 4.9	67.7± 5.3 ●	70.8± 4.6	71.8± 4.9	66.6± 5.1 ●
people	79.5± 4.3	78.4± 4.2	79.6± 4.3	78.7± 4.2	76.0± 4.4 ●

○, ● statistically significant improvement or degradation vs 1-norm SVM

Table 4.5: MIWrapper: Percentage Accuracy for Ensemble Base Learners

Dataset	1-Norm SVM	Random Forest	Adaboost + C4.5	Adaboost + D. Stump	Bagging + C4.5
musk1	83.6±11.7	87.3±10.8	86.8±11.1	84.7±10.7	86.6±11.0
musk2	78.6±13.1	81.0±11.4	82.7±11.8	79.7±10.6	81.5±11.6
eastwest	47.0±28.3	54.0±31.5	56.0±33.5	69.0±26.4	54.0±34.6
westeast	47.5±29.6	55.0±31.4	57.5±34.4	69.0±26.4	53.5±35.0
mutagenesis-atoms	66.5± 2.3	81.9± 8.5 ○	77.4± 8.1 ○	66.5± 2.3	79.7± 8.2 ○
mutagenesis-bonds	66.5± 2.3	83.1± 8.7 ○	82.2± 8.3 ○	73.2± 8.4 ○	82.9± 8.5 ○
mutagenesis-chains	66.7± 2.6	84.2± 7.5 ○	85.1± 7.8 ○	74.0± 7.7 ○	85.3± 7.6 ○
suramin	65.0±45.2	65.0±45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	87.1± 2.4	87.9± 2.6	88.0± 2.6	87.1± 2.4	87.9± 2.6
elephant	83.4± 8.0	87.1± 6.9	84.2± 8.1	85.5± 7.3	84.1± 7.7
fox	58.9± 9.6	64.6± 9.6	62.6± 9.8	65.7± 9.6	65.7± 8.8
tiger	77.7± 8.1	84.3± 8.1 ○	81.0± 8.1	81.8± 8.5	81.7± 9.4
bikes	81.9± 3.9	82.2± 4.2	81.8± 4.5	79.2± 4.6 ●	82.5± 4.0
cars	72.3± 4.9	74.8± 4.6	71.4± 4.9	71.3± 4.8	74.3± 4.5
people	79.5± 4.3	82.6± 4.0 ○	81.5± 4.0	79.5± 4.3	81.8± 3.9 ○

○, ● statistically significant improvement or degradation vs 1-norm SVM

4.2.3 Comparison of Other MI Algorithms

As well as investigating the behaviour of MILES with different base learners, the other goal of this study was to compare MILES to other state-of-the-art multi-instance learning algorithms. Of particular interest is the performance of MILES relative to the other single-instance wrapper algorithms. Before we compare the other MI learning algorithms to MILES, we consider the performance of these other algorithms separately.

MIWrapper

Reasonable classification performance was observed for MIWrapper on all datasets except *eastwest* and *westeast*. The experimental results for MIWrapper with various base classifiers are shown in Tables 4.4 and 4.5.

The non-ensemble base learners for MIWrapper all performed similarly to each other, although the results for C4.5 were significantly superior to the others on the three representations of the mutagenesis problem. The support vector machine with the radial basis function kernel and C4.5 both performed worse than the other non-ensemble base learners on the GRAZ02 image datasets.

All of the ensemble base learners performed better than the 1-norm SVM baseline on the mutagenesis datasets, but there were few other significant differences. Considering that C4.5 was the best performing non-ensemble MIWrapper base learner for mutagenesis, and that all of the ensemble methods used tree-based ensembles, the superiority of the ensemble methods on this dataset may be at least partially due to the decision-tree representations used, rather than the advantages of the ensemble techniques themselves.

The *eastwest* and *westeast* datasets proved to be a challenge for MIWrapper, with the majority of base learners achieving little more than 50% accuracy. The support vector machine with the RBF kernel and Adaboost with decision stumps produced the highest accuracy on these datasets at 68.0% and 69.0%, respectively. The reversal of the class labels between *eastwest* and *westeast* made little difference for MIWrapper.

None of the base learners performed better than the typical result of 65% accuracy on the suramin dataset, and a disastrous result of 35% accuracy was observed for both of the two-norm SVM algorithms. The choice of base learner made virtually no difference for the *thioredoxin* problem, with all MIWrapper classifiers performing at around 87% accuracy.

Summary Statistics for Propositionalization (SimpleMI)

Despite the simplicity of the method, propositionalization by replacing each bag with a feature vector containing the arithmetic averages of the attributes for all of its instances (denoted here as SimpleMI) proved to be very effective on all of the datasets. The results for SimpleMI are presented in Tables 4.6 and 4.7. The 1-norm SVM base classifier was used as a baseline for comparison.

Even more than for MILES, the 1-norm SVM was very competitive as a base learner for SimpleMI. It suffered no significant losses against any other non-ensemble base learner except for C4.5, which achieved a spectacular 95% accuracy on the

Table 4.6: SimpleMI: Percentage Accuracy for Non-Ensemble Base Learners

Dataset	1-Norm SVM	C4.5	SMO (LIN)	SMO (RBF)	Logistic
musk1	81.8±12.0	82.0±14.1	87.2±10.3	76.9±12.8	73.4±13.2
musk2	77.5±12.9	73.8±13.6	81.1±12.0	74.5±12.7	72.2±13.4
eastwest	63.0±33.0	95.0±15.1 ○	57.0±34.1	66.5±31.0	54.5±36.3
westeast	63.0±33.0	95.0±15.1 ○	57.5±33.6	66.5±31.0	54.5±36.3
mutagenesis-atoms	75.5± 9.3	78.9± 9.4	69.5± 7.7 ●	67.3± 8.1 ●	67.5± 8.9 ●
mutagenesis-bonds	84.7± 8.9	81.6± 8.0	74.5± 9.2 ●	73.2± 8.3 ●	83.2± 8.8
mutagenesis-chains	78.8± 8.0	78.9± 8.1	79.4± 7.9	76.8± 9.1	78.5± 8.5
suramin	65.0±45.2	31.5±44.2 ●	74.0±41.7	73.0±42.3	74.0±41.7
thioredoxin	87.0± 2.5	87.0± 2.4	86.5± 4.3	86.7± 3.3	87.6± 4.6
elephant	81.6± 9.5	78.3± 8.6	80.3± 9.8	62.7±11.3 ●	73.3±10.4 ●
fox	54.3±10.1	61.8±11.4	55.9±10.1	57.3± 9.8	55.9±11.3
tiger	79.3± 9.3	76.9± 9.9	79.3± 9.5	71.0±10.0 ●	75.7± 8.5
bikes	81.6± 4.4	76.7± 5.3 ●	83.4± 4.4	75.1± 4.4 ●	81.9± 4.6
cars	71.7± 5.1	68.5± 5.0	70.9± 5.1	65.8± 4.6 ●	71.5± 5.0
people	76.6± 4.6	75.1± 4.2	79.3± 3.9	75.8± 4.6	76.8± 4.7

○, ● statistically significant improvement or degradation vs 1-norm SVM

Table 4.7: SimpleMI: Percentage Accuracy for Ensemble Base Learners

Dataset	1-Norm SVM	Bagging + C4.5	Adaboost + D. Stump	Adaboost + C4.5	Random Forest
musk1	81.8±12.0	81.9±13.8	83.2±12.3	84.5±13.9	84.3±12.1
musk2	77.5±12.9	78.4±11.9	78.7±11.9	77.1±12.9	81.0±11.4
eastwest	63.0±33.0	86.5±23.4 ○	80.0±31.8	80.5±27.4	77.5±27.9
westeast	63.0±33.0	86.5±23.4 ○	81.5±31.5	80.5±27.4	80.0±25.6
mutagenesis-atoms	75.5± 9.3	80.8± 8.1	80.3± 8.4	81.0± 8.6	80.9± 7.6
mutagenesis-bonds	84.7± 8.9	84.8± 8.7	85.8± 7.7	84.7± 8.6	85.8± 8.3
mutagenesis-chains	78.8± 8.0	82.0± 9.2	81.2± 8.8	81.5± 8.2	83.5± 9.0
suramin	65.0±45.2	47.0±48.1	53.0±48.1	47.0±47.6	45.0±47.9
thioredoxin	87.0± 2.5	86.1± 3.5	86.2± 5.3	84.8± 6.2	86.6± 3.7
elephant	81.6± 9.5	83.4± 8.9	86.5± 8.1	84.1± 7.8	87.3± 7.4
fox	54.3±10.1	64.5±10.3 ○	67.0±10.5 ○	63.7± 9.2 ○	63.7± 9.6
tiger	79.3± 9.3	81.1± 9.6	82.5± 8.7	81.5± 8.8	82.9± 8.8
bikes	81.6± 4.4	82.0± 4.8	80.3± 4.9	81.6± 4.6	83.4± 4.0
cars	71.7± 5.1	74.2± 4.5	74.4± 4.4	73.5± 4.1	76.5± 4.2 ○
people	76.6± 4.6	79.1± 4.2	79.0± 4.8	78.9± 5.2	81.5± 3.9 ○

○, ● statistically significant improvement or degradation vs 1-norm SVM

eastwest and *westeast* datasets. Although SimpleMI with the 1-norm SVM achieved no significant wins against the ensemble base learners, it also suffered very few losses. An exception to this is was the *fox* dataset, where the 1-norm SVM base learner only achieved 54.3% accuracy.

Interestingly, the ensemble base learners had virtually identical performances to each other, with a maximum of two significant wins and losses between any given pair of schemes.

Although SimpleMI with bagged C4.5 as the base learner exhibited a nine-percent lower accuracy than when plain C4.5 was used on *eastwest* / *westeast*, this difference was not statistically significant due to the small size of the dataset. Furthermore, the schemes are not directly comparable as the bagged version used an unpruned C4.5 classifier, while pruning was enabled for the plain C4.5.

Table 4.8: Percentage Accuracy For Upgraded Single-Instance and Purpose-Built MI Algorithms

Dataset	EMDD	Citation KNN	MISMO	mi-SVM	Adaboost + Opt.Ball	MILR	MIBoost + REPTREE
musk1	85.2±11.6	83.3±11.2	77.0±13.6	75.9±13.6 •	70.7±13.3 •	73.4±16.1 •	84.5±11.5
musk2	84.7±10.1	75.6±11.3 •	80.2±11.9	71.8±13.8 •	79.5±12.4	77.0±13.2	79.7±11.0
eastwest	68.5±27.2	44.5±35.5	70.0±31.8	61.5±23.4	71.5±30.4	64.0±37.7	59.0±34.4
westeast	31.5±24.3	50.0±31.0	70.0±31.8 ○	39.5±28.7	29.0±24.8	39.5±35.0	61.0±33.0 ○
muta-a	69.6±10.9	72.6± 9.8	67.2± 8.4	66.5± 2.3	72.0± 8.9	72.3± 8.7	77.8± 8.6
muta-b	72.4±12.0	73.3± 8.2	82.5± 9.1 ○	66.5± 2.3	71.9± 9.9	75.8± 9.5	84.4± 7.8 ○
muta-c	70.4±12.5	77.7± 9.7	81.6± 7.9 ○	66.5± 2.3	69.2± 7.5	77.3± 9.3	82.3± 9.0 ○
suramin	49.0±47.7	65.0±45.2	65.0±45.2	65.0±45.2	65.0±45.2	63.0±45.8	65.0±45.2
thiored.	88.5± 5.8	83.5± 5.4	87.1± 2.4	87.1± 2.4	90.3± 5.5	85.4± 5.4	87.3± 4.3
elephant	75.2±10.0	50.0± 0.0 •	81.5± 9.6	78.9± 9.3	72.0±10.3	79.8± 9.0	82.8± 8.6 ○
fox	59.4±10.7	50.0± 0.0 •	53.4±11.5	48.6± 8.2 •	54.6±10.1	59.0±10.5	66.3± 9.5
tiger	73.1±10.1	50.0± 0.0 •	80.5± 8.5 ○	81.5± 9.6	65.2±10.2	75.7± 8.5	82.2± 9.5 ○
bikes	78.3± 6.4	51.0± 0.3 •	81.6± 4.8	83.5± 3.9 ○	78.6± 4.2	82.6± 5.2	80.6± 4.6
cars	72.1± 5.6	52.5± 0.0 •	71.6± 5.2	60.8± 6.3 •	72.2± 4.7	71.5± 4.9	71.5± 5.0
people	77.2± 4.9	55.0± 0.2 •	77.2± 4.5	72.8± 4.5 •	74.4± 4.5	75.0± 5.3	78.9± 4.4

○, • statistically significant improvement or degradation vs EMDD

Purpose-Built and Upgraded Single-Instance Algorithms

The purpose-built and upgraded single-instance algorithms, i.e. the "non-wrapper" MI approaches, were considered together. EMDD, a computationally more efficient refinement of the classic maxDD algorithm, was used as the baseline scheme for the non-wrapper approaches.

It should be noted that the MIBoost algorithm can in fact be viewed as a wrapper, since different single-instance base learners can be selected for boosting. Here, it is viewed as an upgraded single-instance learner. The base learner was selected according to the optimal result found in Dong's comparison of multi-instance learning algorithms [Dong, 2006]. The results for the "non-wrapper" algorithms are summarized in Table 4.8.

The only algorithms to consistently perform better than the baseline EMDD algorithm were the MI kernel algorithm MISMO, and MIBoost. Citation-KNN and the maximum pattern margin support vector machine mi-SVM were statistically worse than EMDD on several datasets. Citation-KNN's poor overall performance was largely due to the image datasets, where it failed to perform significantly better than chance in all cases. Adaboost with Optimal Ball and MILR were statistically even with the baseline on all datasets except *musk1*.

A point of interest is the *westeast* problem, where disastrous performance was observed from all of the algorithms that relied upon the standard MI assumption, despite generally reasonable results on *eastwest* — a problem that is identical except

Table 4.9: Percentage Accuracy of Wrapper Algorithms — 1-Norm SVM Base Learner

Dataset	MILES	SimpleMI	MIWrapper
musk1	83.2±11.7	81.8±12.0	83.6±11.7
musk2	90.4± 9.3	77.5±12.9 •	78.6±13.1 •
eastwest	50.0± 0.0	63.0±33.0	47.0±28.3
westeast	50.0± 0.0	63.0±33.0	47.5±29.6
mutagenesis-atoms	77.4± 9.3	75.5± 9.3	66.5± 2.3 •
mutagenesis-bonds	73.1±13.0	84.7± 8.9 ○	66.5± 2.3
mutagenesis-chains	77.0± 9.0	78.8± 8.0	66.7± 2.6 •
suramin	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	88.1± 5.1	87.0± 2.5	87.1± 2.4
elephant	84.3± 8.7	81.6± 9.5	83.4± 8.0
fox	61.0± 9.5	54.3±10.1	58.9± 9.6
tiger	82.0± 8.5	79.3± 9.3	77.7± 8.1
bikes	76.1± 5.0	81.6± 4.4 ○	81.9± 3.9 ○
cars	66.5± 5.5	71.7± 5.1 ○	72.3± 4.9 ○
people	71.1± 4.9	76.6± 4.6 ○	79.5± 4.3 ○

○, • statistically significant improvement or degradation vs MILES

that the positive and negative bag-labels are reversed. This indicates that *eastwest* is consistent with the standard MI assumption and *westeast* is not [Dong, 2006]. It demonstrates that standard MI assumption-based algorithms are not generally robust against class label reversal.

4.2.4 Comparison of MILES to Other Algorithms

The wrapper algorithms were compared with respect to classification accuracy and training time. Tables 4.9 – 4.14 show the results for the wrapper algorithms with several base learners selected as representative strong schemes: the 1-norm SVM, Adaboost with decision stumps (100 trees) and random forests (100 trees).

Comparison to Other Wrappers: Accuracy

In terms of classification accuracy, MILES exhibited only one significant win against the corresponding SimpleMI classifier over all three base learning schemes. There were no significant differences between MILES and SimpleMI for the boosted decision stumps (Table 4.10), but SimpleMI was superior for the three GRAZ02 datasets using the other two single-instance base classifiers (Tables 4.9 and 4.11).

MILES and MIWrapper were even using the 1-norm SVM base learner, with three significant wins and losses each (Table 4.9). MILES was superior to MIWrapper on the mutagenesis datasets for boosted decision stumps (Table 4.10), while MIWrapper performed better than MILES on three image datasets when using random forests (Table 4.11).

Table 4.10: Percentage Accuracy of Wrapper Algorithms — Adaboost with Decision Stump Base Learner (100 Trees)

Dataset	MILES	SimpleMI	MIWrapper
musk1	88.0±11.6	83.2±12.3	84.7±10.7
musk2	83.2±11.5	78.7±11.9	79.7±10.6
eastwest	81.0±24.4	80.0±31.8	69.0±26.4
westeast	81.0±24.4	81.5±31.5	69.0±26.4
mutagenesis-atoms	83.9± 8.6	80.3± 8.4	66.5± 2.3 •
mutagenesis-bonds	86.3± 7.4	85.8± 7.7	73.2± 8.4 •
mutagenesis-chains	86.0± 8.0	81.2± 8.8	74.0± 7.7 •
suramin	65.0±45.2	53.0±48.1	65.0±45.2
thioredoxin	89.3± 4.0	86.2± 5.3	87.1± 2.4
elephant	80.9± 7.7	86.5± 8.1	85.5± 7.3
fox	61.6±10.9	67.0±10.5	65.7± 9.6
tiger	80.5± 8.9	82.5± 8.7	81.8± 8.5
bikes	78.0± 5.0	80.3± 4.9	79.2± 4.6
cars	71.6± 4.1	74.4± 4.4	71.3± 4.8
people	75.6± 4.6	79.0± 4.8	79.5± 4.3 ○

○, • statistically significant improvement or degradation

Table 4.11: Percentage Accuracy of Wrapper Algorithms — Random Forest Base Learner (100 Trees)

Dataset	MILES	SimpleMI	MIWrapper
musk1	87.0±11.4	84.3±12.1	87.3±10.8
musk2	81.7±11.2	81.0±11.4	81.0±11.4
eastwest	80.0±24.6	77.5±27.9	54.0±31.5
westeast	80.0±24.6	80.0±25.6	55.0±31.4
mutagenesis-atoms	82.0± 8.2	80.9± 7.6	81.9± 8.5
mutagenesis-bonds	79.7±10.5	85.8± 8.3	83.1± 8.7
mutagenesis-chains	80.4± 9.2	83.5± 9.0	84.2± 7.5
suramin	65.0±45.2	45.0±47.9	65.0±45.2
thioredoxin	87.7± 2.7	86.6± 3.7	87.9± 2.6
elephant	82.3± 8.2	87.3± 7.4	87.1± 6.9
fox	64.9±10.2	63.7± 9.6	64.6± 9.6
tiger	78.6± 9.0	82.9± 8.8	84.3± 8.1 ○
bikes	79.2± 4.4	83.4± 4.0 ○	82.2± 4.2 ○
cars	71.7± 4.0	76.5± 4.2 ○	74.8± 4.6
people	77.5± 4.3	81.5± 3.9 ○	82.6± 4.0 ○

○, • statistically significant improvement or degradation vs MILES

Table 4.12: CPU Secs Training Time for Wrapper Algorithms — 1-Norm SVM Base Learner

Dataset	MILES	SimpleMI	MIWrapper
musk1	0.3± 0.0	0.1±0.0 •	1.9± 0.2 ○
musk2	21.2± 3.2	0.1±0.0 •	509.6±135.6 ○
eastwest	0.0± 0.0	0.0±0.0 •	0.0± 0.0
westeast	0.0± 0.0	0.0±0.0 •	0.0± 0.0 ○
mutagenesis-atoms	2.2± 0.2	0.0±0.0 •	0.7± 0.0 •
mutagenesis-bonds	10.2± 0.7	0.0±0.0 •	7.1± 0.6 •
mutagenesis-chains	17.5± 1.7	0.1±0.0 •	16.9± 1.0
suramin	0.4± 0.0	0.0±0.0 •	0.9± 0.1 ○
thioredoxin	86.6± 5.1	0.0±0.0 •	816.4± 41.4 ○
elephant	5.8± 0.5	0.3±0.0 •	18.7± 3.2 ○
fox	9.7± 0.7	0.4±0.0 •	28.2± 2.6 ○
tiger	5.4± 0.4	0.3±0.0 •	16.8± 1.6 ○
bikes	240.5±13.1	4.0±0.3 •	350.7± 14.5 ○
cars	423.9±14.6	6.6±0.4 •	631.3± 33.6 ○
people	219.5± 9.4	3.8±0.2 •	296.6± 10.9 ○

○, • statistically significant degradation or improvement vs MILES

Table 4.13: CPU Secs Training Time for Wrapper Algorithms — Adaboost with Decision Stump Base Learner (100 Trees)

Dataset	MILES	SimpleMI	MIWrapper
musk1	4.3± 0.3	3.0±0.1 •	4.7± 0.2 ○
musk2	284.7±41.5	3.5±0.0 •	151.8±19.0 •
eastwest	0.2± 0.1	0.0±0.0 •	0.2± 0.0
westeast	0.2± 0.1	0.0±0.0 •	0.2± 0.0
mutagenesis-atoms	17.9± 0.2	0.1±0.0 •	0.7± 0.0 •
mutagenesis-bonds	53.5± 0.6	0.2±0.0 •	3.1± 0.1 •
mutagenesis-chains	88.1± 0.8	0.2±0.0 •	10.2± 0.2 •
suramin	3.7± 0.3	0.0±0.0 •	1.7± 0.1 •
thioredoxin	624.2± 6.7	0.1±0.0 •	32.2± 0.4 •
elephant	36.5± 0.4	3.7±0.1 •	15.8± 0.2 •
fox	34.0± 0.4	3.3±0.9 •	14.7± 0.1 •
tiger	30.4± 0.4	1.8±0.3 •	13.7± 0.1 •
bikes	451.2± 1.3	12.3±0.2 •	66.3± 0.3 •
cars	524.7± 0.5	5.2±0.0 •	73.4± 0.3 •
people	385.1± 0.4	4.5±0.0 •	58.7± 0.2 •

○, • statistically significant degradation or improvement vs MILES

Comparison to Other Wrappers: Training Time

SimpleMI always had the shortest training time of the three methods, for all datasets and base learners (Tables 4.12 – 4.14). This is unsurprising, given that the method only generates one instance for each training bag, without increasing the dimensionality of the feature space. Although MILES also generates one instance per training bag, the dimensionality of the feature space is almost always much higher, as the number of attributes is equal to the total number of instances in the training bags. In contrast, MIWrapper generates one instance for every instance in every bag, leaving the dimensionality of the feature space unchanged.

The relative computational efficiency of the training procedures for MILES and MIWrapper depends on both the base learner and the dataset. Tables 4.12 and 4.13 show that MILES was typically faster than MIWrapper with the 1-norm SVM base

Table 4.14: CPU Secs Training Time for Wrapper Algorithms — Random Forest Base Learner (100 Trees)

Dataset	MILES	SimpleMI	MIWrapper
musk1	3.5± 0.2	1.5±0.1 •	7.3± 0.4 ○
musk2	269.1±40.3	1.7±0.1 •	121.5±12.8 •
eastwest	0.3± 0.0	0.1±0.0 •	0.6± 0.0 ○
westeast	0.3± 0.0	0.0±0.0 •	0.6± 0.0 ○
mutagenesis-atoms	27.2± 0.6	0.3±0.0 •	3.4± 0.1 •
mutagenesis-bonds	76.0± 1.6	0.3±0.0 •	11.3± 0.3 •
mutagenesis-chains	110.4± 2.0	0.4±0.0 •	25.6± 0.7 •
suramin	4.9± 0.4	0.0±0.0 •	6.5± 0.4 ○
thioredoxin	472.0± 7.5	0.2±0.0 •	91.2± 2.2 •
elephant	33.5± 0.5	5.3±0.2 •	46.8± 0.9 ○
fox	35.1± 0.5	6.4±0.2 •	53.8± 1.0 ○
tiger	28.2± 0.4	3.7±0.0 •	36.1± 0.7 ○
bikes	427.5± 2.9	11.4±0.2 •	90.3± 0.7 •
cars	538.2± 3.4	9.0±0.1 •	112.7± 1.0 •
people	364.4± 5.9	6.6±0.1 •	97.1±29.1 •

○, • statistically significant degradation or improvement vs MILES

classifier, while MIWrapper was almost always faster when boosted decision stumps were used. When using random forests, the two methods were roughly even overall in terms of statistical wins and losses, but MILES was far slower than MIWrapper on a number of problems, while MIWrapper's losses were generally more modest.

Overall Comparison of Classification Accuracy

To compare the different approaches, the classification accuracy of the best variants of MILES, the purpose-built/upgraded single instance algorithms, MIWrapper and SimpleMI are shown in Table 6.5.

Interestingly, the best results for each type of scheme were seldom more than a few percentage points different from each other. Notable exceptions to this are the *eastwest* / *westeast* datasets, where the best MILES classifier was around ten percentage points ahead of the best MIWrapper classifier and the best non-wrapper scheme, and SimpleMI was fourteen percentage points ahead of the best MILES scheme. There was also a difference of around nine percentage points between the best MILES classifier and the best SimpleMI classifier on the *musk2* dataset, and a difference of around eight percentage points in the case of MILES and MIWrapper. Note that the σ value for MILES used in these experiments was selected by [Chen et al., 2006] based on tuning experiments on a subset of the *musk2* dataset, so the results for MILES on that dataset may be slightly optimistic.

Adaboost with decision stumps was the dominant base learner for MILES, being the best (or best-equal) scheme for eight of the fifteen datasets. MIBoost was the

Table 4.15: The Best Result For Each Type of Scheme

Dataset	Best MILES	%	Best Upgraded/P.B.	%	Best MIWrapper	%	Best SimpleMI	%
musk1	Adaboost + D. Stump	88.0	EMDD	85.2	Random Forest	87.3	SMO (POLY)	87.2
musk2	1-Norm SVM	90.4	EMDD	84.7	SMO (RBF)	82.8	SMO (POLY)	81.1
eastwest	Adaboost + D.Stump	81.0	Adaboost + Opt.Ball	71.5	Adaboost + D. Stump	69.0	C4.5	95.0
westeast	Adaboost + D.Stump	81.0	MISMO	70.0	Adaboost + D. Stump	69.0	C4.5	95.0
mutagenesis-atoms	Adaboost + D. Stump	83.9	MIBOOST + REPTREE	77.8	Random Forest	81.9	Random Forest	80.9
mutagenesis-bonds	Adaboost + D. Stump	86.3	MIBOOST + REPTREE	84.4	Random Forest	83.1	Random* Forest	85.8
mutagenesis-chains	Adaboost + D. Stump	86.0	MIBOOST + REPTREE	82.3	Bagging + C4.5	85.3	Random Forest	83.5
suramin	1-Norm* SVM	65.0	Citation* KNN	65.0	1-Norm* SVM	65.0	SMO* (Poly)	73.0
thioredoxin	Adaboost + D. Stump	89.3	Adaboost + Opt.Ball	90.3	Adaboost + C4.5	88.0	Logistic	87.6
elephant	1-Norm SVM	84.3	MIBOOST + REPTREE	82.8	Random Forest	87.1	Random Forest	87.3
fox	Random Forest	64.9	MIBOOST + REPTREE	66.3	Adaboost* + D. Stump	65.7	Adaboost +D.Stump	67.0
tiger	1-Norm SVM	82.0	MIBOOST + REPTREE	82.2	Random Forest	84.3	Random Forest	82.9
bikes	SMO (LIN)	80.5	mi-SVM	83.5	Bagging + C4.5	82.5	Random* Forest	83.4
cars	SMO (LIN)	72.5	Adaboost + Opt.Ball	72.2	Random Forest	74.8	Random Forest	76.5
people	Random Forest	77.5	MIBOOST + REPTREE	78.9	Random Forest	82.6	Random Forest	81.5

∗ Scheme was best-equal with one or more other schemes.

strongest overall method from the purpose-built / upgraded single-instance category, and the random forests algorithm was the best overall base learner for MIWrapper and SimpleMI.

4.2.5 Conclusions

The goals of the study were to compare base learners for MILES, and to compare MILES to other state-of-the-art MI algorithms. The results indicate that the 1-norm SVM is not generally superior to the standard 2-norm SVM as a base learner for MILES, despite the sparsity property that was claimed to be important for the high-dimensional feature space created by the MILES transformation [Chen et al., 2006]. Moreover, although the 1-norm SVM was a competitive base learner for MILES in the experiment, Adaboost with decision stumps exhibited the strongest performance overall.

The results also show that when appropriate base learners are used, MILES is competitive in classification performance with any purpose-built algorithm or upgraded single-instance learner considered in the experiments. However, the simpler

MIWrapper and SimpleMI methods almost always perform just as well as MILES. Furthermore, it appears that SimpleMI is always significantly superior to MILES in terms of CPU training time. MIWrapper often has shorter training times than MILES, but this depends on the base learner and the dataset used. To achieve good classification accuracy in a wide variety of cases, random forests can be recommended as a base learner for MIWrapper and SimpleMI.

Perhaps the most interesting result of the experiments is the effectiveness of the extremely simple propositionalization methods SimpleMI and MIWrapper in comparison to MILES. The results also confirm their good performance with respect to the sophisticated purpose-built algorithms and upgraded single-instance learners [Dong, 2006]. It appears to be an open problem for the multiple-instance learning community to find "true" MI algorithms that are superior to these simple propositionalization techniques, or to find problems where the existing MI algorithms are more effective than propositionalization.

Chapter 5

New Algorithms and Assumptions for Learning Instance Weights

All multi-instance learning algorithms depend on some particular assumption regarding the relationship between the instances in a bag and that bag's class label [Xu, 2003]. These *MI assumptions* determine the type of multi-instance concepts that can be learnt by algorithms that use those assumptions. Different assumptions are appropriate for different problem domains. For instance, the standard MI assumption is believed to be appropriate for the *musk* problem [Dietterich et al., 1997].

This chapter presents two new MI assumptions that are designed to model the notion of varying levels of influence that instances have on bag-level class labels. One model is based on the collective MI assumption, and the other is inspired by linear classification techniques for single-instance learning. We consider these types of assumptions to be more appropriate for some problem domains than existing MI assumptions, particularly for the domain of content-based image retrieval and other image mining tasks. The assumptions are general enough to be applicable to problems where either the standard MI assumption or the collective assumption hold.

We also describe new "wrapper"-type algorithms to learn MI concepts under these weight-based assumptions. By varying the single-instance base learners for the wrapper algorithms, a wide variety of MI concepts can be learned. The algorithms are evaluated on artificial data in this chapter. A comprehensive evaluation of the new algorithms using real-world data is performed in Chapter 6.

5.1 Motivations for Learning Instance Weights

The goal of the new MI assumptions introduced in this chapter is to model MI concepts where different parts of instance space have different levels of influence on bag-level class labels. The general idea is that each point in instance space "pulls" bag-level classification towards a specific class, and some areas of instance space have a stronger "pull" than others.

This model is motivated by the content-based image retrieval problem (CBIR). Recall that in CBIR, the task is to identify relevant images based on their content. In the typical multiple-instance learning representation of the problem, each image is represented by feature vectors extracted from segments of the image. See Section 2.2.2 for more information on this problem.

The standard MI assumption is not guaranteed to hold for CBIR. For example, consider the task of finding all images of beach landscapes. There is no single item contained in a segment of a beach scene that defines it as belonging to the "beach" category. Instead, we would expect to find a number of items that each contribute to the likelihood that the image is a beach scene, such as sand, ocean, sky, tufts of grass, beach balls, palm trees, sandcastles, and so on.

Also, some unrelated items such as skyscrapers, icebergs or spaceships may decrease the likelihood that an image is a beach scene. Some items influence the class label more than others. For instance, the presence of sand and ocean segments would strongly increase the probability that an image is a beach landscape, but the presence of people in the image would not have a great impact on the class label. Our new MI assumptions are designed to model this type of interaction between instances and bag labels.

Admittedly, the interactions between segments and class labels in real-world CBIR tasks may often be more sophisticated than those that can be represented by the proposed type of model. However, as more powerful assumptions are used, the concept space becomes larger, and thus optimal concept descriptions may be harder to find. Our MI assumptions are designed to be a good compromise on model complexity.

Two artificial problem domains are now provided to illustrate the type of concepts that the new MI assumptions are intended to model. The first domain is called

Trial. Imagine that a person, hereby referred to as X, is on trial for the murder of Y. Suppose that there are a finite number of possible pieces of evidence, denoted $E_i, 1 \leq i \leq n$. There are several pieces of evidence for and against a guilty conviction for X. The obvious question is posed: should X be convicted of the murder?

Clearly, X should only be convicted if the evidence for his or her guilt outweighs the evidence for his innocence. The word "outweighs" gives away the crucial nature of the scenario: the concept of *weight* can be used to model the strength of a piece of evidence towards a given verdict. The weight of each piece of evidence can be predicted via the outcomes of earlier murder cases.

The *Trial* problem can be formulated as an MI problem as follows: A murder case is represented by a bag. The instances within the bags are one-dimensional feature vectors, where the single feature is a nominal attribute with n values designating the presence of a piece of evidence E_i. The task is to determine whether X is guilty for a given murder case, based on a given set of precedent murder cases.

We may assume that each piece of evidence has a weight associated with it determining the relevance of the evidence for or against a guilty verdict. A learning program must predict these weights in order to make accurate predictions. Thus we can see that MI instance weights can be used to model the notion of *relevance of evidence*. Note that this artificial problem is for illustrative purposes only, and the author in no way endorses the use of this model in a real courtroom situation.

The second illustrative problem is called *Scholarship*. In this problem, an ambitious young student has submitted applications for a number of academic scholarships, each with similar levels of competitiveness. For simplicity, we will assume that each scholarship is equally difficult to obtain. Now, each scholarship application is endorsed by several referees, who have each written one reference for the student. To ensure that the referees can be completely candid, the student is not allowed to view any of the references. The referees are all busy academics who do not have time to write different references for each scholarship application, so the student knows that all references from a given referee are identical.

Now, the student is in a difficult situation, as she wants to obtain as many scholarships as possible, but she does not know which references will give her the best chance of success. However, she can attempt to estimate the effectiveness of each reference based on the results of previous applications.

We can formulate the problem as follows: Each scholarship application is represented by a bag. Each instance in a bag corresponds to a reference. The instance space can be described by a single nominal variable, as in the *Trial* case. Each reference influences the outcome of the scholarship application by a specific amount, towards a specific outcome. The learning task is to predict what effect each reference has on the success of the application, in order to make optimal choices of references for future applications.

In this case, it is critical that the learning model outputs instance weight estimates, because the predicted instance weights are more useful than the resulting classifier. Although it is nice to know whether a scholarship application is likely to be successful, even more important is the knowledge of which references are the most beneficial. From this example domain it can be seen that instance weights are useful for modeling the *influence of an instance on an outcome*.

5.2 Upgrading the Collective Assumption To Model Instance Weights

This section presents a generalization of the collective assumption that includes a weight function over instance space. The collective assumption states that each instance in a bag contributes equally and independently to that bag's class label. Recall the class probability equation for the collective assumption:

$$Pr(c|b) = E_X[Pr(c|x)|b] = \int_X Pr(c|x)Pr(x|b)\,dx \ ,$$

where c is a class label, and b is a bag. As stated earlier, in practice $Pr(x|b)$ is estimated from the sample provided by the instances in the bag:

$$Pr(c|b) = \frac{1}{n_b}\sum_{i=1}^{n_b} Pr(c|x_i) \ ,$$

where n_b is the number of instances in b. We now extend this model by incorporating a weight function into the collective assumption:

$$Pr(c|b) = \frac{1}{\sum_{i=1}^{n_b} w(x_i)}\sum_{i=1}^{n_b} w(x_i)pr(c|x_i) \ , \tag{5.1}$$

where $w(x) : \chi \rightarrow \mathbb{R}^+$ is a weight function from instance space to the positive real numbers that determines the level of influence that an instance has on the bag-level class label. This model is called the *weighted collective MI assumption*. The weighted collective assumption asserts that each instance contributes independently *but not necessarily equally* to the class label of the bag.

The weighted collected assumption, as stated in Equation 5.1, is a probabilistic model. Sometimes, however, we may wish to represent a deterministic generative model. This can be achieved within the weighted collective assumption framework by modifying the model to always label a bag with the "most likely" class according to the probability function described in Equation 5.1. In the case of binary classification, this corresponds to:

$$v_{dw}(B) = \text{sign}(t), t = \frac{1}{\sum_{i=1}^{n_b} w(x_i)} \sum_{i=1}^{n_b} w(x_i) pr(+|x_i) - 0.5 \ . \tag{5.2}$$

Here, t is the decision variable, the sign of which determines the classification outcome. The case of $t = 0$ can be decided arbitrarily — we elect to break ties in favour of the positive class, although there is no particular justification for this choice.

This MI assumption (in both the probabilistic and deterministic forms) is more powerful than the collective assumption because it allows some instances to be ignored when determining bag-level class labels. The collective assumption gives all instances in a bag the same weight, which means that every instance must be taken into account, and irrelevant instances may bias the class probability estimates in some problem domains. For instance, the collective assumption cannot model the standard MI assumption, where only a few (positive) examples affect the class labels of bags. Under the weighted collective assumption, we can very closely approximate the standard MI assumption by giving positive instances a large weight and setting all other weights to values close to zero.

For a more concrete example, imagine that we are interested in the CBIR task of finding images of tigers in an image library. The distribution of the image library is skewed, so that the prior probability of an arbitrary image containing a tiger is 85%. A new image contains a segment with a pair of eyes in it. However, let's also imagine that our library consists entirely of images of terrestrial animals, and thus

every image in the library contain eyes.

From the prior probabilities, we know that the probability of an image containing eyes belonging to the class *tiger* is 85%, due to the skewed nature of the database. But the "eyes" concept has no discriminating power, and thus should be disregarded, despite the high probability of a positive example given an instance belonging to that concept. Using the collective assumption we are unable to model this case correctly, as this irrelevant feature must be taken into account.

In another CBIR scenario, suppose that beach scenes are the target concept. Here, the probability that a segment containing an image of a tree belongs to a beach scene is quite low, and the probability that a segment containing an image of some sand belongs to a beach scene is quite high. However, tree segments are almost irrelevant compared to beach segments — many tree segments will not prevent the image from being a beach scene as long as some sand segments are still present. To represent this properly, it is necessary to give tree segments small weights, and sand segments relatively large weights.

The weighted collective assumption can also be used to model the *Trial* and *Scholarship* scenarios. For *Trial*, the class probability function represents the degree of likelihood that X murdered Y given a piece of evidence, and the weight function represents the importance of that piece of evidence on the verdict. For instance, a witness may claim that they saw X commit the murder. This piece of evidence would be associated with a high probability for the *guilty* outcome. However, if there were video camera footage that clearly identifies the murderer as another person, all other evidence would be irrelevant.

In the *Scholarship* scenario, the class probability function represents the degree to which the referee recommends the applicant, and the weight function represents the level of importance that the scholarship selection committee attributes to the reference, which may be determined by the reputation of the referee.

5.3 An Iterative Framework for Learning Instance Weights

When learning predictive models under the weighted collective MI assumption, the model parameters to be learnt are the weight function $w(x)$, and the instance-level class probability function $pr(c|x)$. When these parameters are known, bag-level class probabilities can then be generated using Equation 5.1 (or Equation 5.2).

We propose a heuristic wrapper approach for estimating these functions, based on the MIWrapper algorithm described in Section 3.3.2. The algorithm is called *Iterative Framework for Learning Instance Weights* (IFLIW).

Similarly to MIWrapper, instance-level class labels for training instances are estimated by simply applying the bag-level labels to all instances, and the instance-level class probability function is estimated using a single-instance learning algorithm.

The challenge, however, is to estimate the weight function. An iterative method is applied, where instance weights of the training data are updated according to an update function. The iteration continues until a stopping criterion is met. The weight function is then estimated using a regression model built on the training instance weights.

Possible stopping criteria include predictive performance on the training data or a hold-out validation dataset, or a fixed number of iterations. Exploratory results suggested that a performance-measuring stopping criterion can cause premature termination of the iteration, as performance sometimes drops slightly before continuing to increase. Hence, using a fixed number of iterations is the option used in the remainder of this thesis. The number of iterations is a parameter to the algorithm, and represents a trade-off between computation time and closeness of fit to the data.

In each iteration, the update function modifies the instance weights by multiplying them by $exp(info\_gain(pr(c|x)))$, where $info\_gain$ is the information gain of $pr(c|x)$, the class probability distribution for the instance x predicted by the single-instance base classifier relative to the prior class probabilities computed from the class frequencies in the training data. The instance weights are then normalized, so that the total weight of all of the instances in all of the bags sums to the number of instances. Intuitively, this gives higher weights to the instances whose predicted class probabilities contain the most information.

In order to make use of the updated instance weights, in each iteration the single-instance classifier is rebuilt on a version of the weighted training instances with weights normalized so that each bag has the same total weight, and the sum of all of the instance weights is equal to the number of instances. This "per bag" normalization is performed on a copy of the data so that the instance weights learned by the algorithm remain bag-independent. The only requirement is that the base learner can understand instance weights.

When the weights have been learned, a single-instance regression model is built on the training instances, using the instance weights as the class values. This regression model is used to estimate the weight function $w(x)$. At prediction time, bag-level class probabilities are computed using Equation 5.1, with the single-instance regression model and the single-instance classifier (from the last iteration) used to estimate $w(x)$ and $pr(c|x)$ respectively.

For implementation reasons, the above approach is modified so that the natural logarithms of the instance weights are stored and manipulated, instead of the raw weights themselves. The regression model learns the log-weights instead of the raw weights, and the log-weights are converted to raw weights at prediction time. This helps to improve numeric stability, and also avoids the problems caused by negative values being predicted by some regression algorithms. The pseudocode for the algorithm is provided in Algorithm 3. Entropy is assumed to be computed based on natural logarithms, and all procedure parameters are passed by reference.

5.3.1 Discussion of the Algorithm

IFLIW attempts to learn weighted MI concepts by approximating the instance-level weight function $w(x)$ and class probability function $pr(c|x)$ using a single-instance regression learner R and a single instance classifier L, respectively. In this sense, it is a wrapper algorithm, which makes use of single instance learners to learn MI concepts.

The method for learning $pr(c|x)$ is borrowed from the MIWrapper algorithm. The MI training dataset is transformed into a single-instance training set for classifier L by appending bag-level class labels to all of the instances in all of the training bags. This is a heuristic solution that necessarily introduces bias, but is often a good

Algorithm 3 IFLIW

D = the set of training bags
L = a single-instance base learner
R = a single-instance regression learner
μ = the number of iterations, a parameter to the algorithm

train(D)
C = all instances in the bags in D, labeled with bag-level class labels
$classPriors[]$ = the bag-level prior probabilities for the classes, based on D
for (each instance $i \in C$) **do**
    $i.lnWeight = 0$ // corresponds to a true weight of 1
$normalizeWeightsPerBagAndLearnClassifier(C, L)$
for $(numIterations = 0, numIterations < \mu, numIterations{+}{+})$ **do**
    $updateWeightsInLogSpace(C, L)$
    $normalizeWeightsPerBagAndLearnClassifier(C, L)$
F = copy of C with class of each instance i set to $i.lnWeight$ and weight set to 1
$R.learnRegressionModel(F)$

$normalizeWeightsPerBagAndLearnClassifier(C, L)$
E = a copy of C
for (each $i \in E$) **do**
    $oldBagWeights[i.parentBagID] \mathrel{+}= \exp(i.lnWeight)$
for (each $i \in E$) **do**
    $i.weight = \exp\big(\ln(\frac{|C|}{|D|}) - \ln(oldBagWeights[i.parentBagID]) + i.lnWeight\big)$
$L.train(E)$

$updateWeightsInLogSpace(C, L, classPriors[])$
for (each instance $i \in C$) **do**
    $probs[] = L.predictClassProbabilities(i)$
    $update = entropy(classPriors) - entropy(probs)$ // calc info gain
    $i.lnWeight = i.lnWeight() + update$ // because this is in log space, the true weight is effectively multiplied by $\exp(info\_gain)$
$oldSum = \sum_{i \in C} \exp(i.lnWeight)$
for (each instance $i \in C$) **do**
    $i.lnWeight = \ln(|C|) - \ln\big(oldSum\big) + i.lnWeight$ // normalize log weights

$classify(B)$, $B = \{x_i : i = 1, \cdots, |B|\}$ a test bag
for (each instance $x_i \in B$) **do**
    $lnWeight = R.regressionPrediction(x_i)$
    $instProbs[] = L.predictClassProbabilities(x_i)$
    for $(j = 0, j < numClasses; j{+}{+})$ **do**
        $probs[j] \mathrel{+}= \exp(lnWeight) \times instprobs[j]$
$oldSum = \sum_i probs[i]$
for $(j = 0, j < numClasses; j{+}{+})$ **do**
    $probs[j] = \frac{1}{oldSum} \times probs[j]$ // normalize probs to sum to 1
return *probs*

enough approximation to allow even the simple MIWrapper algorithm to achieve high accuracy predictions on real-world data [Xu, 2003]. The main benefits of this approach are simplicity and computational efficiency.

The weight function is initially learnt only for the instances in the training bags. The entire weight function $w(x)$ is represented by a regression model trained on the instances in the training bags. Choosing an appropriate regression base learner allows the algorithm to represent a wide variety of weight functions. The search for the weight function is then reduced to the search for appropriate weights on the training instances.

The resulting search space is generally very large, as the dimensionality grows linearly with the number of instances in the training bags. In general, a brute-force approach is out of the question. For the sake of computational feasibility, a heuristic approach is required to find a good assignment of instance weights to model the weight function.

The iterative procedure that IFLIW uses for finding instance weights is effectively a guided heuristic search. In each iteration, the instances whose predicted class probabilities give the most information relative to the prior probabilities are given higher weights, and vice-versa. Normalization is performed to help keep the weights in a reasonable numeric range. The new weights are taken into account in each iteration by rebuilding the base classifier using a copy of the instances with weights normalized so that each bag has the same total weight.

In this way, as the iterative process continues, the instances that are the most useful in prediction are given large weight values, and instances that do not contribute to classification are given small weight values. If the base learner is powerful, there is a risk of overfitting to the data when too many iterations are performed. The number of iterations, μ, can be varied to allow for an appropriate closeness of fit to the data. Note that IFLIW with $\mu = 0$ is equivalent to the MIWrapper algorithm.

5.3.2 Computational Complexity of IFLIW

In this section, we informally show an upper bound on the computational complexity of IFLIW's build and classification routines with respect to the number of

instances in the training bags, given some fairly unrestrictive assumptions. Here, the dimensionality of the feature space and the number of classes are both viewed as constants — i.e., we assume that the computational complexities of all base learners and subroutines can be described in terms of the number of instances in the bags in the training set.

Theorem 5.3.1. *Let n = the number of instances in all of the bags in D = $|C|$ = $|E|$ = $|F|$, $L.train(E) \in O(l(n))$, $R.train(F) \in O(r(n))$ and $L.predictClassProbabilities(x) \in O(p(n))$ Assume that L and R use all of the data, i.e. $l(n), r(n) \in \Omega(n)$. Then $IFLIW.train(D) \in O(\mu(np(n) + l(n)) + r(n))$.*

Proof. Collecting the instances in all of the bags and labeling them with bag level class labels is clearly $O(n)$. Similarly for computing the class priors. The initialization of the log weights is also $O(n)$. Temporarily ignoring the core part of the algorithm, the creation of F is $O(n)$. The regression model must be constructed in the last step — this is $O(r(n))$. Each of these operations occurs once, so disregarding $normalizeWeightsPerBagAndLearnClassifier(C, L)$ and the mainfor loop, the algorithm is

$$
\begin{aligned}
& O(n + n + n + n + r(n)) \\
& \in O(4n + r(n)) \\
& \in O(n + r(n)) \\
& \in O(r(n)) \text{ (since } r(n) \in \Omega(n)).
\end{aligned}
$$

We will now compute the complexity of the routine $normalizeWeightsPerBagAndLearnClassifier(C, L)$. The creation of E, a copy of C, is $O(n)$. Assuming that parent bag IDs have been cached and can be computed in $O(1)$ time, the two for loops in the routine are both $O(n)$. By definition, $L.train(C) \in O(l(n))$. So the $normalizeWeightsPerBagAndLearnClassifier(C, L)$ routine is

$$
\begin{aligned}
& O(n+n+n+l(n)) \\
& \in O(3n+l(n)) \\
& \in O(n+l(n) \\
& \in O(l(n)) \text{ (since } l(n) \in \Omega(n))
\end{aligned}
$$

Next, we compute the complexity of $updateWeightsInLogSpace(C, L)$. $L.predictClassProbabilities()$ is $O(p(n))$ by definition. The computation of entropy is $O(1)$ under our assumption that the number of classes is a constant. The update of the log weight is also $O(1)$. The first loop iterates n times, and so is $O(n(p(n) + 2) \in O(np(n))$. The computation of $oldSum$ is $O(n)$, and so is the normalization loop. So the $updateWeightsInLogSpace(C, L)$ routine is $O(np(n) + 2n) \in O(n(p(n) + 2)) \in O(np(n))$.

Returning to the $train()$ procedure, the main for loop iterates μ times, and each iteration is $O(l(n)+np(n)$. Therefore the for loop is $O(\mu(l(n)+np(n)))$. Combining the loop and the remainder of the algorithm, $ILFIW.train(D) \in$

$$
\begin{aligned}
& O(\mu(np(n)+l(n))+l(n)+r(n)) \\
& \in O((\mu+1)(np(n)+l(n))+r(n)) \\
& \in O(\mu(np(n)+l(n))+r(n)) \ .
\end{aligned}
$$

□

For example, consider the case where the single-instance classification base learner is $C4.5$ without using subtree replacement for pruning, and the regression base learner is additive regression with decision stumps. The computational cost $l(n)$ of building a C4.5 decision tree without subtree replacement is is $O(n \log n)$ [Witten and Frank, 2005]. If we assume (as Witten and Frank do) that the depth of the tree is $O(\log n)$, the complexity of the classification rou-

tine $p(n)$ must also be $O(\log n)$. The cost of the creation of a decision stump is $O(n \log n)$, and additive regression merely multiplies this by a constant factor, so $r(n) \in O(n \log n)$. So with these base learners, the complexity of $IFLIW.train(D)$ is $O(\mu(n \log(n) + n \log n) + n \log n) \in O(\mu n \log n)$.

Theorem 5.3.2. *Let $R.regressionPrediction(x) \in O(e(n))$, B be a bag, and $|B| = m$. Then $IFLIW.classify(B) \in O(m(e(n) + p(n)))$.*

Proof. The first for loop executes m times. In each iteration of that loop, a regression prediction and a classification prediction are made; these are $O(e(n))$ and $O(p(n))$, respectively. As the number of classes is viewed as a constant, the loop over the classes is $O(1)$. So the first for loop is $O(m(e(n) + p(n)))$.

The computation of $oldSum$ and the normalization loop are $O(numClasses) \in O(1)$ by assumption, and the $return$ statement is also $O(1)$. Therefore, the $classify()$ procedure is

$$\begin{aligned} &O(m(e(n) + p(n)) + 3) \\ &\in O(m(e(n) + p(n))) \end{aligned}$$

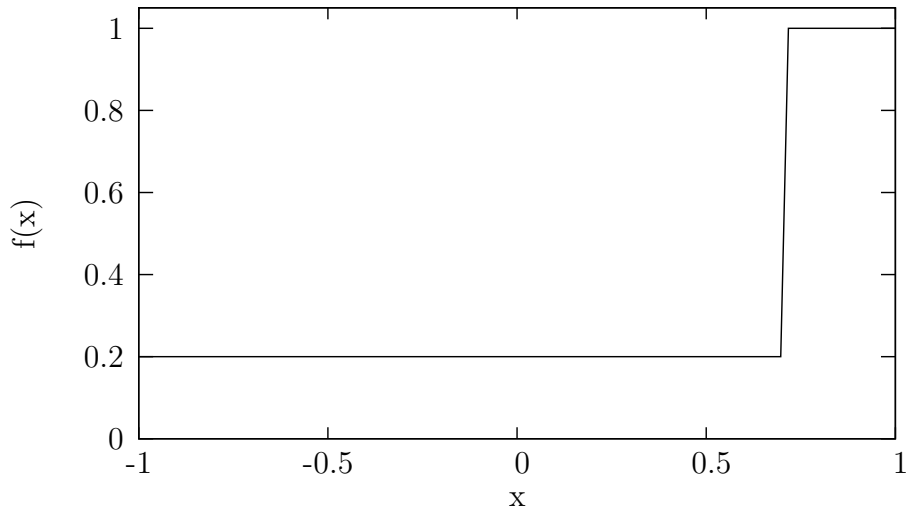
□

Continuing with the example scenario used to illustrate the previous result, we know that the complexity of the classification routine for a decision stump is $O(1)$. Additive regression multiplies this by a constant factor which has no impact on the *big-Oh* bounds for the routine, so we have $e(n) \in O(1)$. Then, the complexity of IFLIW's classification routine using C4.5 (without subtree replacement) and additive regression with decision stumps is $O(m(1 + \log n)) \in O(m \log n)$.

5.3.3 Evaluation on Artificial Data

In order to investigate the effectiveness of the algorithm, 64 artificial datasets were generated using the weighted collective assumption. The instance space for each dataset consisted of a single numeric attribute $x \in [-1, 1]$. The class probability function $pr(x)$ and the weight function $w(x)$ were varied over the following four functions:

Figure 5.1: Artifical Function 0 — Decision Stump



- *decision stump*: $f(x) = \begin{cases} 1 & x = 0.7 \\ 0.2 & \text{otherwise} \end{cases}$

- *piecewise (decision tree)*: $f(x) = \begin{cases} 1 & x < -0.5 \\ 0.2 & -0.5 \leq x < 0 \\ 1 & 0 \leq x < 0.5 \\ 0.5 & \text{otherwise} \end{cases}$

- *linear* $f(x) = 0.3x + 0.5$

- *sigmoid* $f(x) = 1 - (1/(1 + exp(10x)))$

Graphs for each of the functions are provided in Figures 5.1 to 5.4. Also varied were the determinism of the class labels and the size of the bags. In the case of deterministic classification, the class label of each bag was set to the most likely class according to Equation 5.2. Otherwise, bag-level class labels were assigned randomly according to the computed bag-level probability distribution described by Equation 5.1. When bags sizes were deterministic, each bag contained exactly five instances, otherwise the size of each bag was randomly selected between one and five instances. Each dataset contained 150 bags. Varying these parameters, a total of $4 \times 4 \times 2 \times 2 = 64$ different datasets were generated.

Figure 5.2: Artifical Function 1 — Piecewise (Decision Tree)

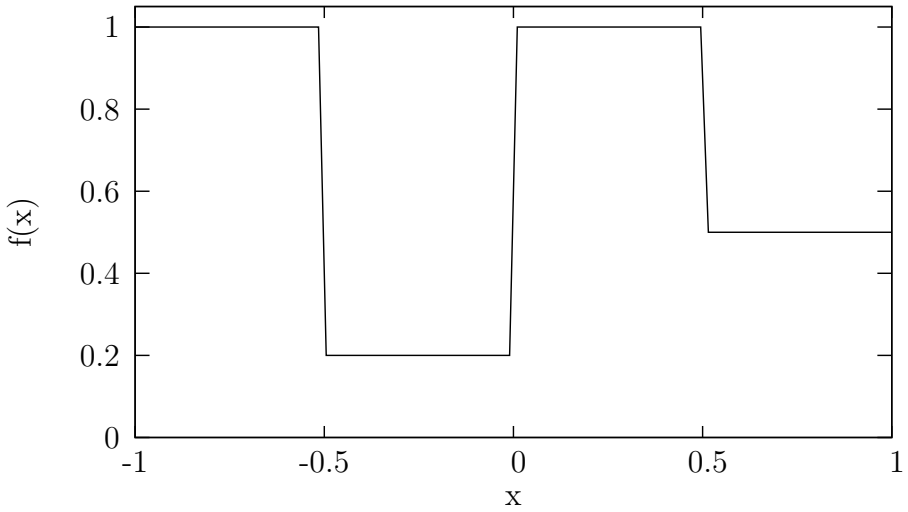


Figure 5.3: Artifical Function 2 — Linear

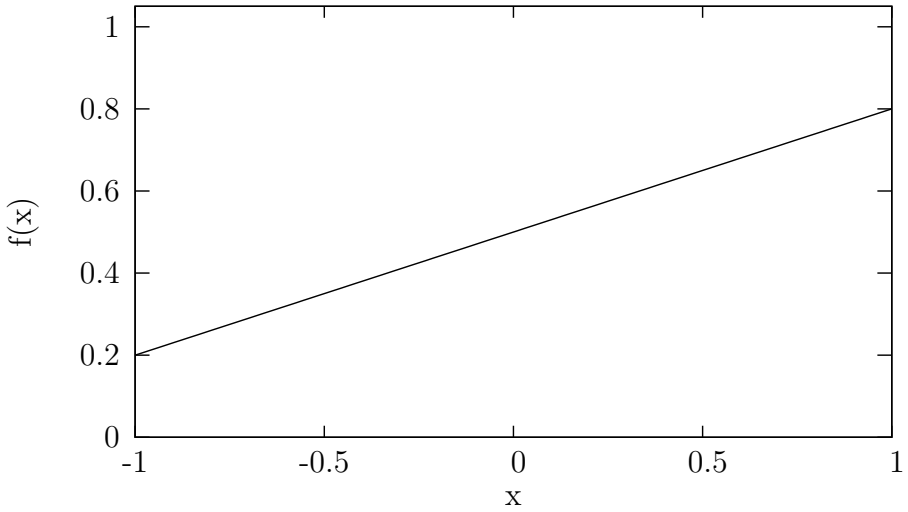
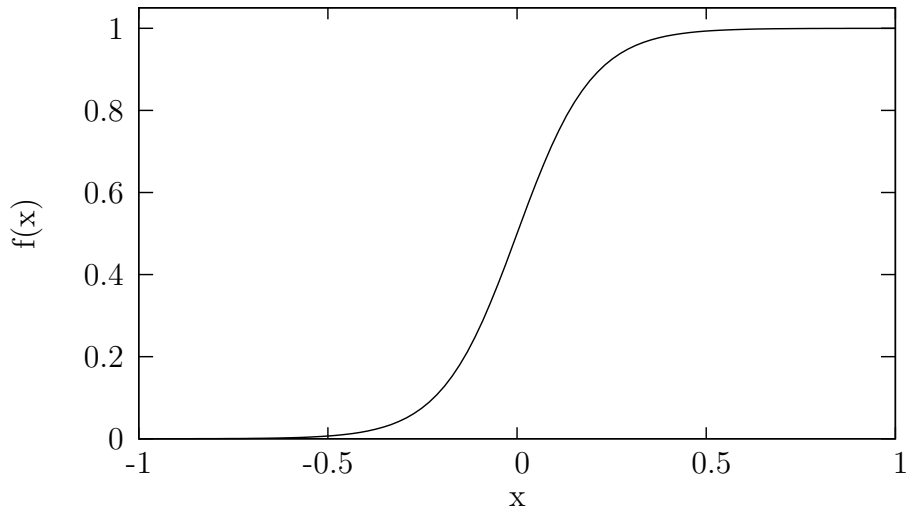


Figure 5.4: Artifical Function 3 — Sigmoid



IFLIW was evaluated on the artificial datasets using the single-instance base learners from the MILES experiment. Additive regression with 100 decision stumps was used as the regression learner. The number of iterations for IFLIW was set to 20. Ten repeats of 10-fold cross validation were performed on each dataset using each base learner, and statistical significance was determined using a pairwise resampled corrected t-test with significance level $\alpha = 0.05$. The MIWrapper algorithm was used as a baseline in the experiments.

Apart from classification accuracy, the root mean squared error (RMSE) was used to measure the quality of the probability estimates generated. The RMSE is defined as: $\sqrt{\sum_{ij}(p_{ij} - a_{ij})^2/n}$, where for all test bags $B_i \in \{B_1, \cdots, B_n\}$ and for all class labels $c_j \in \{c_1, \cdots, c_m\}$, p_{ij} is the predicted probability that B_i belongs to class c_j, and $a_{ij} = 1$ if and only if B_i belongs to class c_j, otherwise $a_{ij} = 0$.

Because of the squaring of the error terms, a property of the root mean squared error measure is that it gives more weight to large errors than to small ones. The results for IFLIW relative to MIWrapper for the two performance measures are summarized in Tables 5.1 and 5.2.

When executing the experiments, a difficulty was encountered with the random forests algorithm when used as a base learner for both the IFLIW and MIWrapper MI algorithms. On some datasets, trees of such great depth were created that the learning program crashed due to a lack of stack space. Although the WEKA implementation of the random forests algorithm is widely used and well tested, it

Table 5.1: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	5	3	3	8	3	5	6	34	6
Losses	4	3	3	6	3	8	3	0	4
Ties	55	58	58	50	58	51	55	30	54
Wins - Losses	1	0	0	2	0	-3	3	34	2
Average MIWrapper	75.1	78.8	68.0	67.1	68.0	77.9	76.2	63.0	77.9
Average IFLIW	74.9	78.0	68.0	67.2	68.0	77.4	76.5	72.5	77.8

Table 5.2: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	24	28	28	29	28	27	35	33	33
Losses	1	1	1	3	1	2	0	9	0
Ties	39	35	35	32	35	35	29	22	31
Wins - Losses	23	27	27	26	27	25	35	24	33
Average MIWrapper	0.43	0.46	0.46	0.47	0.46	0.43	0.43	0.47	0.43
Average IFLIW	0.38	0.44	0.44	0.45	0.44	0.36	0.35	0.41	0.36

seemed likely that this was caused by a bug in the implementation of that algorithm. This problem was resolved by limiting the depth of the trees created by the random forests algorithm to 500 levels — a depth which was not expected to be reached in normal execution of the algorithm on these datasets.

It was found that although IFLIW did not typically achieve superior classification accuracy over the baseline MIWrapper algorithm (Table 5.1), it frequently obtained significantly lower root mean squared error rates compared to MIWrapper (Table 5.2) for all base learners, with very few significant losses for this performance measure. The median number of significant wins minus significant losses against MIWrapper was only one for classification accuracy, but 27 for the root mean squared error measure.

The root mean squared error is a more sensitive measure of performance than classification accuracy, which may partially explain the discrepancy between the results of the two measures. Furthermore, as found by [Dong, 2006] and confirmed in the experiments described in Chapter 4, MIWrapper is typically very difficult to beat in terms of classification accuracy. It appears that the bias introduced by MIWrapper's inability to represent instance weights is small enough to make little difference to classification performance (due to an adequate decision boundary being obtained), but the bias in the probability estimates becomes apparent when the root

Table 5.3: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Deterministic Generative Model (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	5	3	3	7	3	5	6	31	6
Losses	4	3	3	4	3	8	3	0	4
Ties	23	26	26	21	26	19	23	1	22
Wins - Losses	1	0	0	3	0	-3	3	31	2
Average MIWrapper	88.7	76.3	76.3	74.9	76.3	91.7	90.9	71.3	91.4
Average IFLIW	88.5	76.3	76.3	75.1	76.3	90.4	90.1	87.0	91.1

Table 5.4: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Non-Deterministic Generative Model (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	0	0	0	1	0	0	0	3	0
Losses	0	0	0	2	0	0	0	0	0
Ties	32	32	32	29	32	32	32	29	32
Wins - Losses	0	0	0	-1	0	0	0	3	0
Average MIWrapper	61.5	59.7	59.7	59.3	59.7	64.0	61.4	54.6	64.4
Average IFLIW	61.3	59.7	59.7	59.2	59.7	64.4	62.0	58.0	64.5

Table 5.5: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Deterministic Generative Model (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	21	23	23	24	23	26	28	31	29
Losses	0	1	1	2	1	0	0	0	0
Ties	11	8	8	6	8	6	4	1	3
Wins - Losses	21	22	22	22	22	26	28	31	29
Average MIWrapper	0.39	0.44	0.44	0.45	0.44	0.38	0.39	0.43	0.38
Average IFLIW	0.28	0.39	0.39	0.41	0.39	0.24	0.24	0.28	0.24

Table 5.6: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Non-Deterministic Generative Model (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	3	5	5	5	5	1	7	2	4
Losses	1	0	0	1	0	2	0	9	0
Ties	28	27	27	26	27	29	25	21	28
Wins - Losses	2	5	5	4	5	-1	7	-7	4
Average MIWrapper	0.48	0.48	0.48	0.49	0.49	0.48	0.48	0.52	0.48
Average IFLIW	0.47	0.48	0.48	0.49	0.49	0.48	0.47	0.54	0.47

Table 5.7: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Fixed Bag Sizes (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	2	3	3	5	3	2	2	16	2
Losses	4	2	2	4	2	6	3	0	4
Ties	26	27	27	23	27	24	27	16	26
Wins - Losses	-2	1	1	1	1	-4	-1	16	-2
Average MIWrapper	73.4	66.4	66.4	66.3	66.4	77.0	74.6	60.5	77.4
Average IFLIW	72.3	66.6	66.7	66.4	66.6	75.6	73.7	69.5	75.8

Table 5.8: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Random Bag Sizes (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	3	0	0	3	0	3	4	18	4
Losses	0	1	1	2	1	2	0	0	0
Ties	29	31	31	27	31	27	28	14	28
Wins - Losses	3	-1	-1	1	-1	1	4	18	4
Average MIWrapper	76.8	69.6	69.6	67.8	69.6	78.7	77.8	65.4	78.3
Average IFLIW	77.5	69.4	69.4	68.0	69.4	79.2	79.3	75.5	79.8

Table 5.9: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Fixed Bag Sizes (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	11	14	14	16	14	15	19	15	17
Losses	0	1	1	2	1	0	0	5	0
Ties	21	17	17	14	17	17	13	12	15
Wins - Losses	11	13	13	14	13	15	19	10	17
Average MIWrapper	0.46	0.48	0.48	0.47	0.48	0.45	0.46	0.48	0.45
Average IFLIW	0.40	0.44	0.44	0.45	0.44	0.38	0.38	0.43	0.37

Table 5.10: IFLIW: Significant Wins and Losses vs MIWrapper on Artificial Data with Random Bag Sizes (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	13	14	14	13	14	12	16	18	16
Losses	1	0	0	1	0	2	0	4	0
Ties	18	18	18	18	18	18	16	10	16
Wins - Losses	12	14	14	12	14	10	16	14	16
Average MIWrapper	0.41	0.45	0.45	0.47	0.45	0.41	0.41	0.47	0.41
Average IFLIW	0.35	0.43	0.43	0.45	0.43	0.34	0.33	0.40	0.34

mean squared error measure is used.

A very interesting case was the random forests base learner, for which IFLIW achieved 34 significant wins and no significant losses against MIWrapper for classification accuracy (Table 5.1). This is a surprising result, given that the for next-best base learner (Adaboost with C4.5), it achieved only six significant wins with three losses versus MIWrapper.

However, considering the root mean squared error, the results for random forests are not quite as positive. Although IFLIW with random forests achieved the second-highest number of significant wins versus MIWrapper (with 33 wins), it also had the biggest number of significant losses by quite a large margin. It suffered nine significant losses against MIWrapper, while the next highest number of losses was from the 1-norm support vector machine with only three significant losses. In comparison to this, for six of the nine base learners IFLIW suffered no more than one loss against MIWrapper for the root mean squared error measure.

Closer analysis shows that IFLIW with the random forests base learner consistently performed better than MIWrapper on the datasets that were generated via a deterministic labeling process, but did not exhibit such a clear superiority on the probabilistically labeled datasets.

With the random forests base learner, significant wins for IFLIW were observed for all but one of the deterministically labeled datasets (where no significant difference was observed), under both the classification accuracy (Table 5.3) and root mean squared error performance measures (Table 5.5). In contrast to this, only three wins with respect to accuracy were observed for probabilistically labeled datasets, also with no losses against MIWrapper (Table 5.4). The nine significant losses with respect to the root mean squared error measure for IFLIW with random forests were all observed on probabilistically labeled datasets, while only two significant wins were observed for IFLIW with that base learner for that performance measure (Table 5.6).

Regardless of the base learner used, the majority of cases where IFLIW performed better than MIWrapper were on datasets where a deterministic generative model was used. Overall, IFLIW achieved a total of 223 more significant wins than losses with respect to the root mean squared error on datasets where a deterministic generative model was used (Table 5.5), compared to 24 more wins than losses for that measure

on datasets with probabilistic generative models (Table 5.6).

For the classification accuracy measure, 37 more significant wins than losses for IFLIW were observed on datasets where deterministic generative models were used (Table 5.3), while only two more significant wins than losses were observed on datasets where probabilistic generative models were used (Table 5.4). However, this result was almost entirely due to the random forests base learner.

It was found that both IFLIW and MIWrapper consistently performed slightly better on datasets where bag sizes were randomly selected than on datasets where each bag contained exactly five instances, regardless of the base classifier or performance measure (Tables 5.7 — 5.10). Although this result is slightly counter-intuitive, it can be largely explained by the fact that the datasets where bag sizes were randomly selected contained smaller bags on average, which presumably resulted in an easier learning problem. In hindsight, to make this comparison fairer the mean of the randomly generated bag sizes could have been set to the size of the bags used in the deterministic case. This should be taken into account when interpreting the experimental results with respect to determinism of bag sizes.

In the observed results, and under classification accuracy, IFLIW performed better relative to MIWrapper on datasets with non-deterministic bag sizes than on those with fixed bag sizes, although bag size determinism had a smaller effect on the algorithms' relative performance than label determinism. IFLIW achieved 28 more significant wins than losses against MIWrapper on datasets with randomly selected bag sizes (Table 5.8), compared to 11 more significant wins than losses on datasets with fixed bag sizes (Table 5.7).

In contrast to the case of label determinism, there was very little overall difference between datasets with fixed bag sizes and those with random sizes under the root mean squared error performance measure: 125 more significant wins than losses for IFLIW were observed on datasets with fixed bag sizes (Table 5.9), compared to 122 on the randomly sized datasets (Table 5.10).

Predicted Weights vs Iterations

Figures 5.5 to 5.10 show the weights predicted by IFLIW for the instances in the training bags in their initial state and after each of the first five iterations for one of the artificial problems described above. The weight function and class probability

Figure 5.5: Weight Function — Predicted vs Actual. Iteration 0

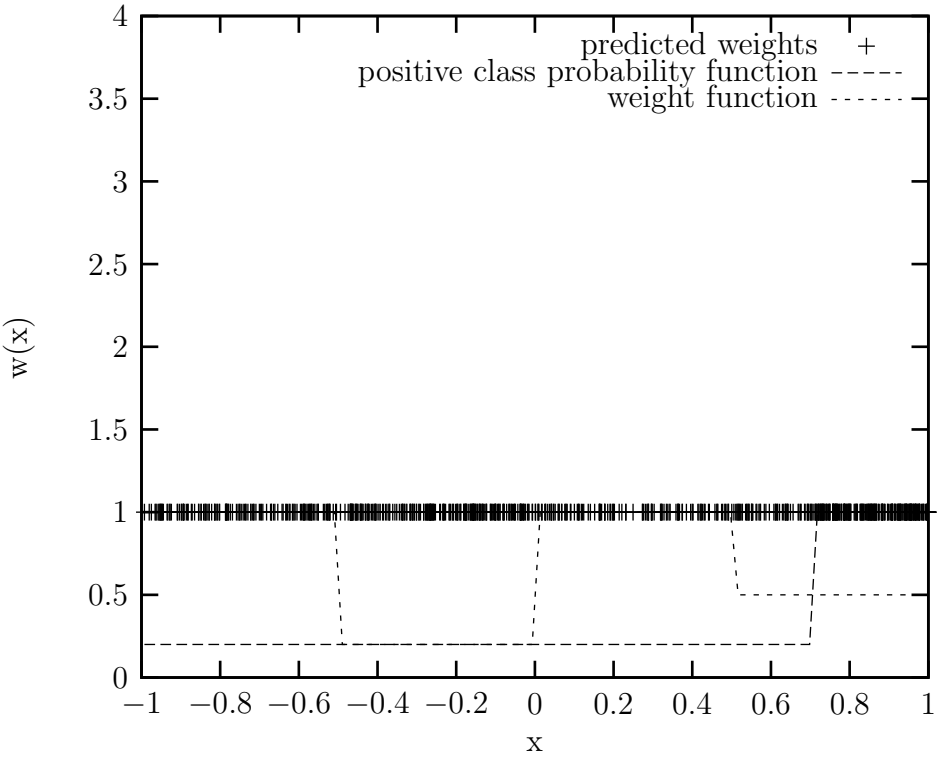


Figure 5.6: Weight Function — Predicted vs Actual. Iteration 1

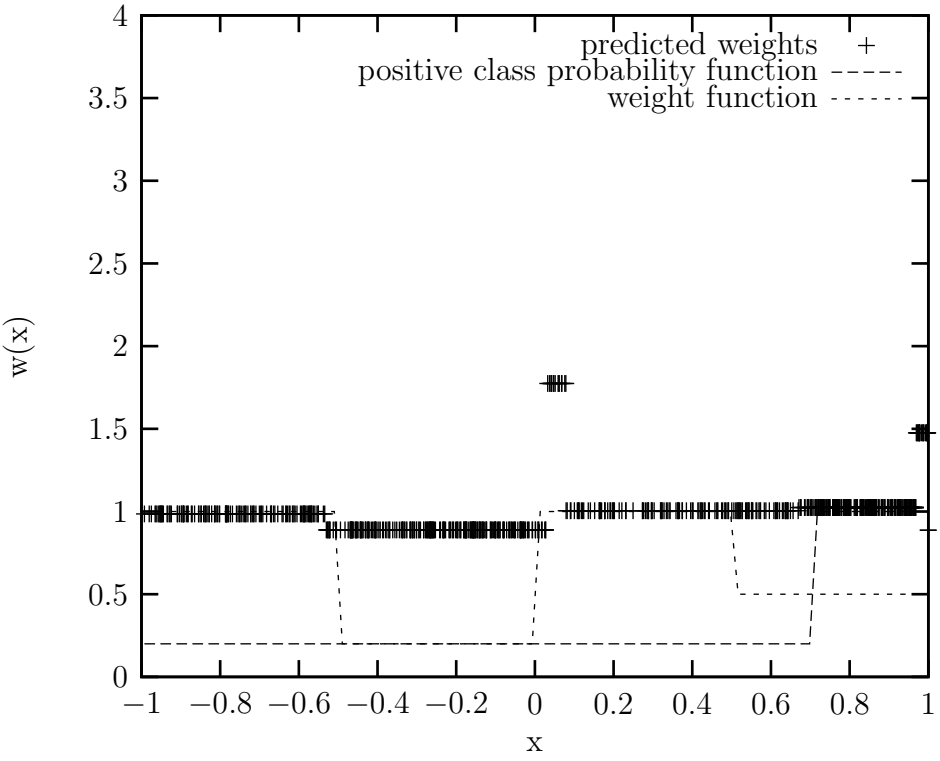


Figure 5.7: Weight Function — Predicted vs Actual. Iteration 2

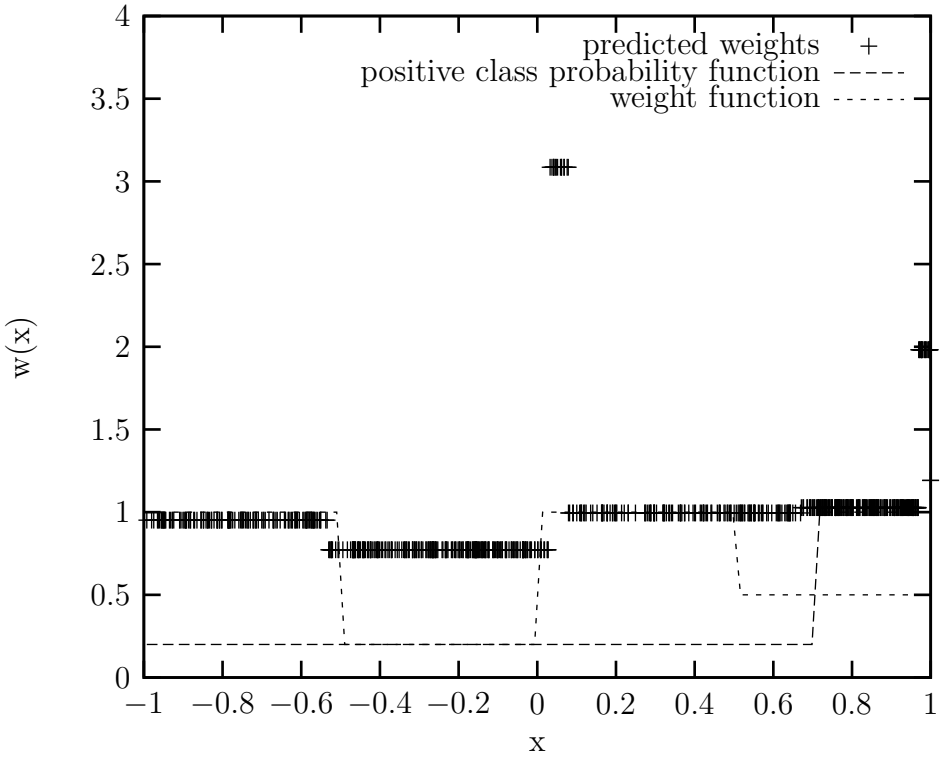


Figure 5.8: Weight Function — Predicted vs Actual. Iteration 3

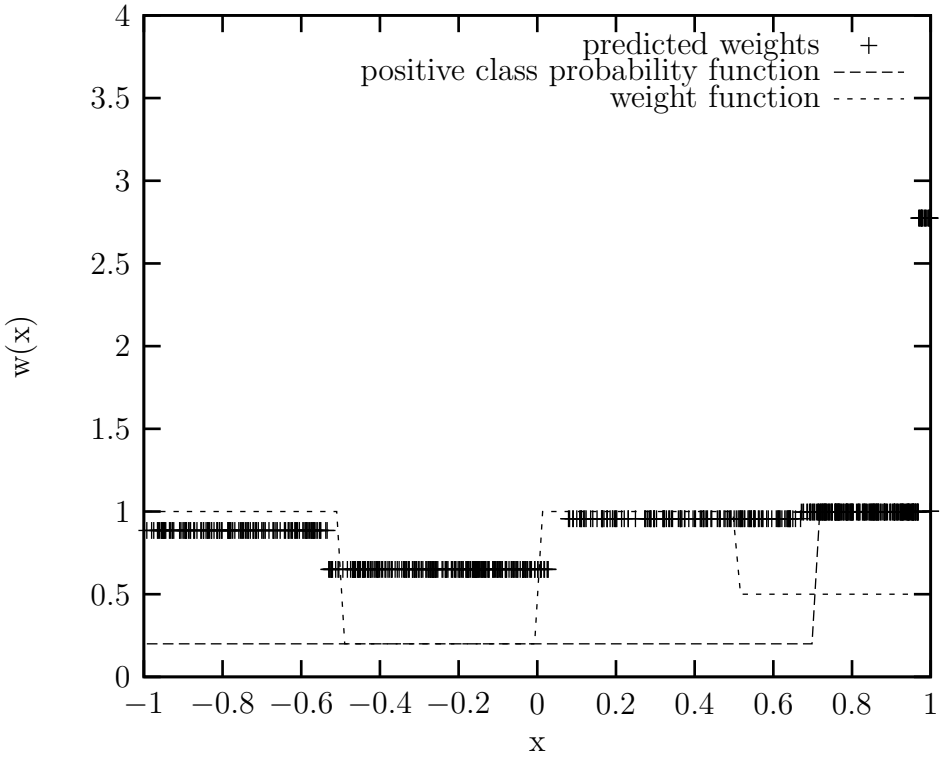


Figure 5.9: Weight Function — Predicted vs Actual. Iteration 4

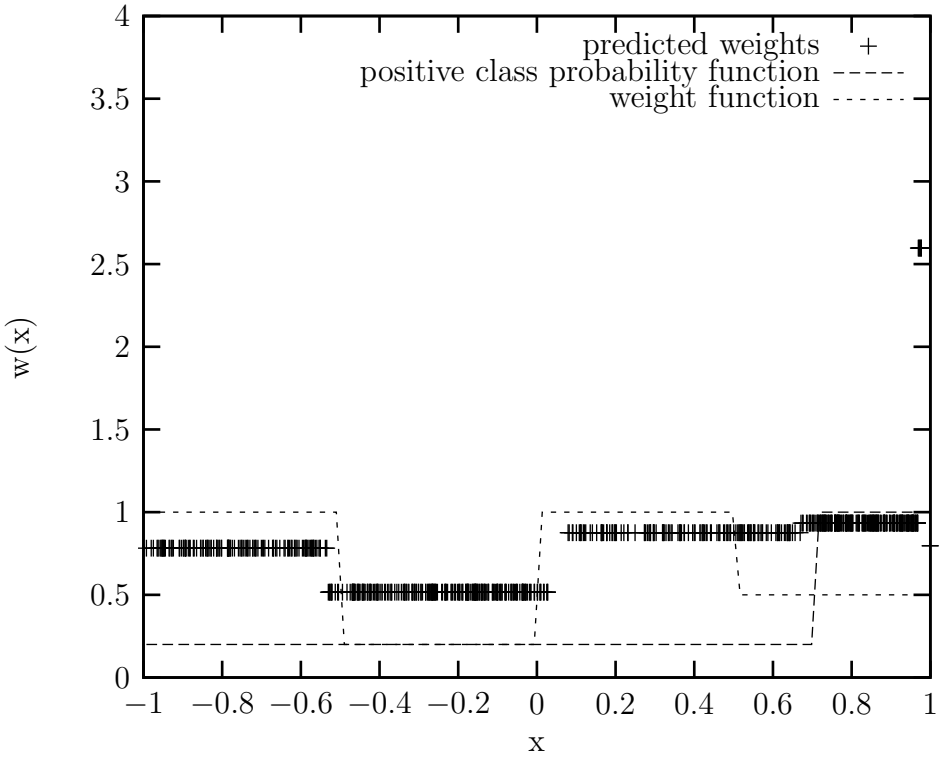
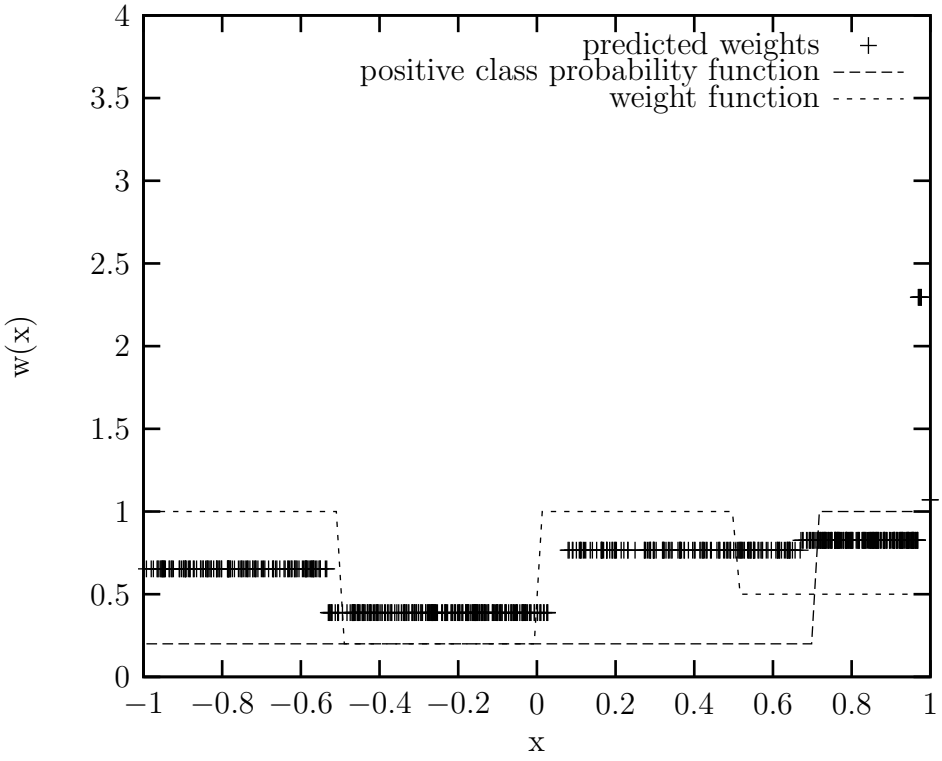


Figure 5.10: Weight Function — Predicted vs Actual. Iteration 5



function used to generate the dataset were the *piecewise (decision tree)* function from Figure 5.2 and the *decision stump* function from Figure 5.1, respectively. The deterministic version of the weighted MI assumption (Equation 5.2) was used to generate 150 bags, each containing five instances.

The single-instance base classifier used for IFLIW was a simple information-gain based decision tree algorithm, implemented as *REPTree* in the WEKA data mining software suite. No pruning was performed for the decision tree, but instead the tree was limited to a maximum depth of three levels. The choice of regression learner is irrelevant here, as the regression model is not built until after the iteration process is completed.

It can be seen from the graphs that in this example, IFLIW has in most cases shifted the instance weights towards the correct shape of the underlying weight function as the iterations have proceeded. However, some issues can also be observed.

Instances belonging to the region of instance space between $x = 0.05$ and $x = 0.1$ were given much higher instance weights than those in other regions. In fact, the weights of points in this region were out of range of the graphs in Figures 5.8 to 5.10. The most likely cause for this is overfitting to the training data. Under this hypothesis, the base learner produced very confident probability estimates in this region due to the fortuitous occurrence of class purity in this region of the training set. The confident probability estimates caused an increase in weight, which was exaggerated in every iteration. By iteration 3 (Figure 5.8), the weights for the instances close to $x = 0.05$ exceeded the maximum value represented by the graph, and by iteration 5 had weight values of approximately 15, a twenty-fold increase over the weight values of nearby instances.

Thus it appears that overfitting by the base learner can be exaggerated by IFLIW, especially when large numbers of iterations are used. For this reason, it is recommended that the base learner for IFLIW be selected carefully to avoid overfitting.

Another interesting phenomenon that can be observed from the graphs is the behaviour of IFLIW in the region where x is between 0.7 and 1. In the underlying model, this region has a moderate weight and a high probability of the positive class. IFLIW gave this region a higher weight than the preceding region, despite the fact that the preceeding region has a higher weight value in the underlying model.

This can be explained by the ambiguity between the weight function and the probability function. In the weighted collective assumption, the total influence of an instance on a bag-level class label is determined not only by the weight value of that instance, but also the confidence of the probability function of the instance. The higher weight in this region was most likely predicted due to interference from the probability function.

It is clear from this example that IFLIW cannot always retrieve the original weight function and probability function. In fact, this is not possible in general, as different combinations of weight and probability function could generate the same overall classification model. Despite this, it does appear to make a reasonable approximation as to which areas of instance space are the most important. From this point of view, the increased weight in the region near the right-hand end of the graph is sensible, as the probability function in that region makes those instances more important than other regions, even though the weight value of that region is not very high in the generative model.

5.4 An Alternative Weighted Assumption

In this section, we present a second MI assumption for modeling the notion of varying levels of influence on bag-level class labels, called the *weighted linear threshold MI assumption*. This assumption is inspired by linear classification techniques such as support vector machines and boosting. In the following, the model is formalized, motivations for the model are discussed, and the relationship between the weighted collective assumption and this alternative assumption is explained. In Section 5.5, we show how MILES can be modified to learn this alternative type of MI concept.

5.4.1 The Weighted Linear Threshold MI Assumption

The weighted collective MI assumption (described earlier in Section 5.2) is a generalized MI assumption that extends the collective assumption. We showed that such an extended model is more powerful than the collective assumption, and that it can also be used to approximate the standard MI assumption.

The relationship between the weighted assumption and the collective assumption

is convenient for upgrading collective assumption-type MI algorithms to incorporate an influence function. However, if the goal is to represent the notion of bag-level influence over instance space, the weighted collective MI assumption may be unnecessarily complex.

Weighted collective MI concepts are defined by a probability function $pr(c|x)$ and a weight function $w(x)$. Under this assumption, the level of influence of a point in instance space depends not only on the weight of the point, but also on the confidence of the class probability. For example, in the deterministic version of the assumption, if there is an equal likelihood that an instance x belongs to either the positive or the negative class ($pr(+|x) = pr(-|x) = 0.5$), x will have no influence on the class label of a bag, regardless of its weight.

In the weighted collective MI model, the weight function and the probability function combine to form an overall influence function. It is this "true" influence function that determines the bag-level classification outcome.

It is not hard to see that many different combinations of probability function and weight function can produce the same overall influence function. Because of this ambiguity, it is not generally possible to recover the true underlying weight and probability functions from an observed set of data. Fortunately, it is not necessary to do so, as the overall influence function is all that is required to make accurate classifications.

Thus, if we are interested only in classification performance, it makes sense to use the notion of instance weights to model bag-level influence directly. We can simplify the weighted collective model by equating the notion of *weight* with the overall influence that an instance has on a bag's class label. In terms of the weighted collective assumption, this new type of weight, combined with the class label of an instance, corresponds to the combined effect of both the level of confidence of $pr(c|x)$, and $w(x)$.

This alternative assumption is named the *weighted linear threshold* MI assumption, because the accumulated (signed) weights for a bag are compared against a threshold to obtain a classification. In this model, a weight function $w_{wlt}(x) : \chi \rightarrow \mathbb{R}^+$ and a classification function $c_{wlt}(x) : \chi \rightarrow \{+1, -1\}$ are defined over instance space[1]. Instances belonging to the positive class ($c_{wlt}(x) = +1$) influence their par-

[1]Here, R^+ refers to the positive real numbers, and does not include zero.

ent bag towards a positive class label, and instances belonging to the negative class ($c_{wlt}(x) = -1$) influence their bag towards a negative class label. The weight of an instance determines the strength of that instance's influence on bag-level class labels.

Formally, let $\nu_{wlt} : \mathbb{N}^{\chi} \rightarrow \Omega = \{+, -\}$ be a weighted linear threshold MI concept. Then ν_{wlt} is of the form

$$\nu_{wlt}(X) = \text{sign}\left(\sum_i w_{wlt}(X_i) c_{wlt}(X_i) + b \right) . \tag{5.3}$$

Here, b is a bias variable, which determines the location of the decision boundary. This formulation of weight-based MI learning is inspired by linear classification in single instance learning (see Section 3.3.5). Recall the classification equation for a linear classifier:

$$\begin{aligned} \nu(m) &= \text{sign}(w^T m + b) \\ &= \text{sign}(\sum_i w_i m_i + b) \ . \end{aligned}$$

In the weighted linear threshold model, instances are treated analogously to attributes in the case of linear classification. The class $c_{wlt}(x)$ of an instance corresponds to an attribute value m_i. Instance weights $w_{wlt}(x)$ in the MI assumption correspond directly to attribute weights w_i in a linear classifier. The bias parameter b performs an identical function to the b parameter in the linear classification model.

The weighted linear threshold assumption can represent the types of concepts that the deterministic version of the weighted collective assumption can model. The precise relationship between the two models is described in Section 5.4.3.

As mentioned earlier, the weighted collective MI assumption presents a difficulty for learning algorithms in that there is an ambiguity between the weight function and the class probability function. Many different pairs of weight and probability function can produce the same overall influence function, which means that the original functions cannot be retrieved in general.

This ambiguity problem is largely avoided with the weighted linear threshold assumption, as the level of influence is determined by a single function, $w_{wlt}(x)$. Be-

cause the classification function encapsulates only the direction of the influence and does not contain information relating to its magnitude, there is no shared responsibility between $w_{wlt}(x)$ and $c_{wlt}(x)$ for the level of influence at a point in instance space.

Some ambiguity in the model remains, however, as different combinations of $w_{wlt}(x)$ and b can correspond to the same classification decision boundary. We will not prove this formally here, but a geometric interpretation may help to demonstrate this. Recall that in this model, instances are treated similarly to the way attributes are treated in linear classification. In other words, each instance can be interpreted as a dimension of a feature space.

Under this interpretation, a weighted linear threshold concept can be viewed as a "hyperplane" through the space of instance labels. The part of Equation 5.3 that describes the location of the decision boundary, $\sum_i w_{wlt}(X_i)c_{wlt}(X_i)+b$, is analogous to a "line" in this space. Just as a line $L = \{m|w^T m + b = 0\}$ in a vector space can be described by different choices of w^T and b, a weighted linear threshold can be described by different choices of $w_{wlt}(x)$ and b.

This ambiguity is less problematic than in the case of the weighted collective assumption, as the ambiguity does not affect the semantics of the model. In the case of the line L, all choices of w^T and b must be scalar multiples of each other — similarly, $w_{wlt}(x)$ and b are unique up to scalar multiplication.

5.4.2 Artificial Domain Examples

To illustrate the type of MI concepts that can be represented by the weighted linear threshold MI assumption, let us consider a simple artificial domain example, which we will refer to as $WLT1$. This artificial problem consists of an instance space χ with one binary attribute $a = \{t, f\}$. As each instance is described entirely by a single attribute value, we will abuse the feature vector notation and denote an instance in this domain by its attribute value, i.e. $\chi = \{t, f\}$.

In our artificial example, let the influence function $w_{WLT1}(x)$ be:

$$w_{WLT1}(x) = \begin{cases} 2 & \text{if } x = t \\ 1 & \text{if } x = f \end{cases},$$

let the classification function be

$$c_{WLT1}(x) = \begin{cases} +1 & \text{if } x = t \\ -1 & \text{if } x = f \ , \end{cases}$$

and set $b = -3$. In this scenario, t is a positive instance, and f is a negative instance. t instances have twice as much influence as f instances. Then the artificial concept v_{WLT1} can be described by:

$$\nu_{WLT1}(X) = \text{sign}\left(2 \times \mathit{count}(X_i = t) - \mathit{count}(X_i = f) - 3\right) \ ,$$

where $\mathit{count}(X_i = x)$ is the number of instances X_i in the bag X where the attribute value for X_i is equal to x.

The *trial* and *scholarship* scenarios can also be represented using the weighted linear threshold assumption. In the *trial* problem, the classification function determines whether a piece of evidence supports a guilty or innocent verdict, and the weight function determines the influence that the piece of evidence has on the verdict.

In the *scholarship* scenario, the classification function determines whether the reference was positive or negative, and the weight function corresponds to the level of influence that the reference has on the scholarship selection committee, which is affected by both the level of enthusiasm of the referee and their reputation.

5.4.3 Relationship to the Weighted Collective MI Assumption

It is instructive to consider how the weighted collective MI assumption is related to the weighted linear threshold assumption. We show the relationship between the two models by deriving a formula to convert an arbitrary deterministic weighted collective concept into a weighted linear threshold concept, thus demonstrating that the weighted linear threshold assumption is at least as powerful a concept language as the deterministic weighted collective assumption. A partial converse is also shown.

Theorem 5.4.1. *Let $\nu_{dw}(B)$ be a deterministic weighted collective concept. Then $\nu_{dw}(B)$ can be written as a weighted linear threshold concept of the form:*

$$\nu_{dw}(B) = sign\Big(\sum_i w_{dw}(x_i)c_{dw}(x_i) + b\Big)$$

Proof. The deterministic weighted collective concept $v_{dw}(B)$ is of the form:

$$\nu_{dw}(B) = \text{sign}(t), t = \frac{1}{\sum_{j=1}^{n_b} w(x_j)} \sum_{i=1}^{n_b} w(x_i)pr(+|x_i) - 0.5$$

The target variable t determines the classification of a bag. To convert $v_{dw}(B)$ into a weighted linear threshold concept, we will find a method to compute the class of an instance x_i, and the influence that x_i has on t. First, we rewrite the equation as

$$\frac{w(x_1)pr(+|x_1) + \cdots + w(x_{n_b})pr(+|x_{n_b})}{\sum_{j=1}^{n_b} w(x_j)} = t + 0.5 \ .$$

Let $a_i = pr(+|x_i) - 0.5$. Then we have

$$\begin{aligned}
&\frac{w(x_1)(a_1 + 0.5) + \cdots + w(x_{n_b})(a_{n_b} + 0.5)}{\sum_{j=1}^{n_b} w(x_j)} = t + 0.5 \\
\Leftrightarrow\ & w(x_1)(a_1 + 0.5) + \cdots + w(x_{n_b})(a_{n_b} + 0.5) = \sum_{j=1}^{n_b} w(x_j)(t + 0.5) \\
\Leftrightarrow\ & w(x_1)(a_1 + 0.5) + \cdots + w(x_{n_b})(a_{n_b} + 0.5) = \sum_{j=1}^{n_b} w(x_j)t + \sum_{j=1}^{n_b} 0.5w(x_j) \ .
\end{aligned}$$

Expanding the left hand size and collecting like terms into sums, we get

$$\sum_{i=1}^{n_b} w(x_i)a_i + \sum_{i=1}^{n_b} 0.5w(x_i) = \sum_{j=1}^{n_b} w(x_j)t + \sum_{j=1}^{n_b} 0.5w(x_j)$$

$$\Leftrightarrow \quad \sum_{i=1}^{n_b} w(x_i)a_i = \sum_{j=1}^{n_b} w(x_j)t$$

$$\Leftrightarrow \quad \frac{\sum_{i=1}^{n_b} w(x_i)a_i}{\sum_{j=1}^{n_b} w(x_j)} = t$$

$$\Leftrightarrow \quad \frac{\sum_{i=1}^{n_b} w(x_i)(pr(+|x_i) - 0.5)}{\sum_{j=1}^{n_b} w(x_j)} = t$$

$$\Leftrightarrow \quad \sum_{i=1}^{n_b} \frac{w(x_i)(pr(+|x_i) - 0.5)}{\sum_{j=1}^{n_b} w(x_j)} = t$$

We know that $\nu_{dw}(B) = \text{sign}(t)$, so we have:

$$\nu_{dw}(B) = \text{sign}\left(\sum_{i=1}^{n_b} \frac{w(x_i)(pr(+|x_i) - 0.5)}{\sum_{j=1}^{n_b} w(x_j)} \right) .$$

Let $c_{dw}(x) = \begin{cases} +1 & pr(+|x) - 0.5 > 0 \\ -1 & otherwise \end{cases}$,
and let $w_{dw}(x) = |w(x)(pr(+|x) - 0.5)|$. Then

$$\nu_{dw}(B) = \text{sign}\left(\sum_{i=1}^{n_b} \frac{|w(x_i)(pr(+|x_i) - 0.5)|c_{dw}(x_i)}{\sum_{j=1}^{n_b} w(x_j)} \right)$$

$$\Leftrightarrow \nu_{dw}(B) = \text{sign}\left(\sum_{i=1}^{n_b} \frac{w_{dw}(x_i)c_{dw}(x_i)}{\sum_{j=1}^{n_b} w(x_j)} \right)$$

Since $\sum_{j=1}^{n_b} w(x_j)$ is positive and affects all terms equally, it does not affect the sign. Therefore, it can be discarded, and ν_{dw} can be written in weighted linear threshold form:

$$\nu_{dw}(B) = \text{sign}\left(\sum_{i} w_{dw}(x_i)c_{dw}(x_i) + 0 \right) .$$

□

We have now shown that any arbitrary deterministic weighted collective concept can be represented as a weighted linear threshold concept. As every step of the conversion was an equivalence, the converse is true with one restriction: any weighted linear threshold concept where $b = 0$ can be represented as a weighted collective concept.

This makes sense given that there is no bias parameter in the deterministic weighted collective assumption — instead, a fixed threshold of 0.5 is used. To remedy this, a bias parameter b_{edw} can be introduced as follows:

$$\nu_{edw}(B) = \text{sign}(t), t = \frac{1}{\sum_{j=1}^{n_b} w(x_j)} \sum_{i=1}^{n_b} w(x_i) pr(+|x_i) - b_{edw} \ . \tag{5.4}$$

We name this model the *extended deterministic weighted collective assumption*. This thesis will not explore this model any further.

5.5 Modifying MILES to Learn Weighted Linear Threshold Concepts

Although MILES learns instance weights for all of the instances in its training bags, in generalizing these weights to other instances a bag-dependent weight function is used. Because the weight function is bag-dependent, it is not a function over instance space. In other words, MILES' weight method is a function $w_{MILES}(x) : (\chi \times \mathbb{B}) \rightarrow \mathbb{R}$ from the Cartesian product of instance space χ with bag space $\mathbb{B}$ to the real numbers (see Section 3.3.5). It is desirable for the weight method to be a function over instance space, i.e. of the form $w(x) : \chi \rightarrow \mathbb{R}$. This section presents YARDS, a variation on MILES which learns a true weight function over instance space, and makes classifications according to the weighted linear threshold MI assumption.

Recall that MILES performs a transformation that maps every bag into an instance-based feature space, where each feature represents the similarity between a bag and an instance. A single-instance classifier (typically a 1-norm support vector machine) is learned on this transformed feature space, and used for classification.

The similarity measure $s(x, B)$ used for the feature mapping was defined earlier

in Equation 3.2. For convenience, we provide it again:

$$s(x, B) = \max_j \exp\left(- \frac{\|B_j - x\|^2}{\sigma^2} \right) ,$$

where x is an instance, B is a bag and σ is a parameter to the model. As explained above, this is problematic when viewing MILES as an algorithm that learns instance weights, as the weight of any point in instance space is dependent on the other instances in its bag. The difficulty arises because of the max operator, which selects only the closest instance in the bag B when determining the similarity value. The max operator is based on the *most-likely cause estimator* [Maron, 1998], a method from the diverse density framework for predicting the probability that a point in instance space is the target point (assuming only a single target point) given a bag. The most likely cause estimator only considers the closest instance in the bag when computing that probability.

We propose an alternative similarity measure $s_y(X, B)$ that removes this bag dependence. We call this alternative *Yet Another Radial Distance-based Similarity measure* (YARDS). The YARDS similarity measure replaces the max operator with a sum operator. In other words, every instance in the bag contributes in an equal fashion when computing the similarity measure. This results in the following equation:

$$s_y(x, B) = \sum_j \exp\left(- \frac{\|B_j - x\|^2}{\sigma^2} \right) . \tag{5.5}$$

By summing the distance-based similarity measures for each instance in the bag, the weight of each instance is taken into account.

The YARDS algorithm is otherwise identical to MILES. The feature space mapping is performed using the YARDS transformation, which is identical to the MILES transformation except that the alternative similarity measure is used:

$$m(B) = [s_y(x^1, B), s_y(x^2, B), \cdots, s_y(x^n, B)]^T . \tag{5.6}$$

Just as for MILES, a single instance classifier is trained on the transformed feature space, and subsequently used for classification on the similarly transformed test instances. However, unlike MILES, if a linear classifier is used for the single

instance model, a weight function over instance space $w_y(x) : \chi \rightarrow \mathbb{R}^+$ is implicitly formed, and the classifications made by the algorithm are in accordance with the simplified weighted MI assumption. A proof for this is now provided.

Theorem 5.5.1. *YARDS classifiers with linear base learners adhere to the simplified weighted MI assumption. I.e.* $\nu_{YARDS}(X)$ *can be written in the form:* $sign\Big(\sum_i w(X_i)c(X_i) + b\Big)$, *where* $w(x)$ *is a function over the instance space* χ.

Proof. Let us consider the behaviour of the linear model on this transformed data. Linear classification models are of the form:

$$\begin{aligned} \nu(m) &= \text{sign}(w^T m + b) \\ \Leftrightarrow \nu(m) &= \text{sign}(\sum_k w_k m_k + b) \ , \end{aligned}$$

where w is a weight vector, m is a test instance, b is the bias parameter, and $\nu(m)$ is the output of the model for instance m. Let X be the bag that generated instance m. By Equation 5.6, the YARDS classifier can be represented by

$$\begin{aligned} \nu_{YARDS}(X) &= \text{sign}(\sum_k w_k s_y(x^k, X) + b) \\ \Leftrightarrow \nu_{YARDS}(X) &= \text{sign}(\sum_k w_k \sum_j \exp\Big(- \frac{\|X_j - x^k\|^2}{\sigma^2}\Big) + b) \\ \Leftrightarrow \nu_{YARDS}(X) &= \text{sign}(\sum_k \sum_j \exp\Big(- \frac{\|X_j - x^k\|^2}{\sigma^2}\Big) w_k + b) \\ \Leftrightarrow \nu_{YARDS}(X) &= \text{sign}(\sum_j \sum_k \exp\Big(- \frac{\|X_j - x^k\|^2}{\sigma^2}\Big) w_k + b) \ . \end{aligned}$$

We now define $w_y(x) : \chi \rightarrow \mathbb{R}^+$ as follows:

$$w_y(x) = |\sum_k \exp\Big(- \frac{\|x - x^k\|^2}{\sigma^2}\Big) w_k| \ . \tag{5.7}$$

We also let $c_y(x) = \begin{cases} +1 & \sum_k \exp\Big(- \frac{\|x - x^k\|^2}{\sigma^2}\Big) w_k > 0 \\ -1 & otherwise \ . \end{cases}$

Now,

$$\begin{aligned}
&\sum_k \exp\left(-\frac{\|X_j - x^k\|^2}{\sigma^2}\right) w_k \\
&= |\sum_k \exp\left(-\frac{\|X_j - x^k\|^2}{\sigma^2}\right) w_k | c_y(X_j) \\
&= w_j(X_j) c_y(X_j) \ ,
\end{aligned}$$

so we have:

$$\nu_{YARDS}(X) = \text{sign}(\sum_j w_y(X_j) c_y(X_j) + b) \ . \tag{5.8}$$

Assuming a fixed σ, and a fixed set of target points $x^k \in C$ with fixed weights w_k, $w_y(x)$ is clearly a function over instance space χ, and by Equation 5.8 the YARDS classifier adheres to the simplified weighted MI assumption. □

Pseudocode for YARDS is provided in Algorithm 4.

Interpretation of YARDS' weight function

Let us consider Equation 5.7. The weight of an instance at point x is determined by the sum of a set of functions, each of which is related to the distance between x and an instance from a training bag.

We can interpret this by considering each training point to have its own influence function over χ. These influence functions are *Gaussian*, i.e. "bell-shaped" functions, with a peak (or trough, if w_k is negative) at the location of the corresponding training point. The influence functions "stack" together additively to form the overall influence function $w_y(x)$. Thus, the weight function used by YARDS consists of the sum of a set of Gaussian influence functions, each centred at an instance from the training data. It is easy to see that this is quite a powerful representation, and many complex weight functions can be represented in this manner.

5.5.1 Computational Complexity of YARDS

In this section, we informally show upper bounds on the computational complexity of YARDS. It should be noted that the complexity results for YARDS are also

Algorithm 4 YARDS

D = the set of training bags
C = all instances in the bags in D
L = a single-instance base learner
σ = the scaling factor, a parameter to the algorithm

$\underline{YARDS\_transform(B)}$, $B = \{B_j : j = 1, \cdots, n_b\}$, a bag
for (every instance x^k in C) **do**
    $sum = 0$
    for (every instance B_j in B) **do**
        $sum = sum + e^{-\frac{\|B_j - x^k\|^2}{\sigma^2}}$
    the kth element of $m(B)$ is $s_y(x^k, B) = sum$
return $m(B)$

$\underline{train(D)}$
F = an empty set of instances
for (every bag $B_i = \{x_{ij} : j = 1, \cdots, n_i\}$ in D) **do**
    $t = YARDS\_transform(B_i)$
    $t.setClassLabel(B_i.getClassLabel())$
    $F = F \cup \{t\}$
$L.train(F)$ // Can optionally perform feature selection here also

$\underline{classify(B)}$, $B = \{B_j : j = 1, \cdots, n_b\}$ a test bag
$t = YARDS\_transform(B)$
return $L.classify(t)$

applicable to MILES, as the $\sum$ operation used in the YARDS feature space transformation is of the same complexity as the max operation used for MILES, and the algorithms are otherwise identical.

In the following proofs, we assume that the dimensionality of the instance space χ and the number of classes are both constants. All capital letter variable names refer to the pseudocode in Algorithm 4. Further, let $n = |C|$, the number of instances in all of the training bags in D, and let $q = |D|$, the number of training bags.

Theorem 5.5.2. *Let B be a bag, $|B| = m$. Then $YARDS\_transform(B)$, the transformation that maps B into the instance-based feature space, is $O(nm)$.*

Proof. To perform the mapping, a feature is created for each instance in the training bags. As there are n of these instances, the feature creation step occurs n times. The feature value corresponding to each instance x^k is equal to $s_y(x^k, B)$. This is computed by summing the values of a Gaussian function for each instance in B. As we view the dimensionality of the instance space as a constant, the computation of the Gaussian function for each instance is $O(1)$. This occurs m times. So the mapping procedure is $O(n \times m \times 1) \in O(nm)$. □

Theorem 5.5.3. *Let $L.train(F) \in O(f(a, b))$, where a is the number of instances in F and b is their dimensionality. Then $YARDS.train(D) \in O(n^2 + f(q, n))$.*

Proof. Because $YARDS\_transform()$ is called for each training bag, the Gaussian function is computed for each pair of training instance from these bags (of which there are n) and candidate target point (of which there also are n). The computation of the Gaussian function is $O(1)$ by assumption, so the total cost for the Gaussian step over all iterations is $O(1 \times n \times n) \in O(n^2)$.

There are two further $O(1)$ assignment statements in the loop over the candidate target points $x^k \in C$. As that loop repeats n times for each bag, the overall cost of the execution of those statements is $O(nq) \in O(n^2)$. The assignment of class labels to transformed instances occurs once for each bag, and the total cost for this is $O(q) \in O(n^2)$. So the execution of the feature space transformation for all instances is $O(n^2)$.

All that remains is the complexity of the base learner L's $train()$ procedure. An instance in the transformed dataset F is created for each bag in the training data,

so $|F| = |D| = q$. Each instance in F contains a feature corresponding to each instance in C so the dimensionality of F is $|C| = n$. So $L.train(F) \in O(f(q,n))$. Putting the two steps together, the whole build procedure is $O(n^2 + f(q,n))$. □

As an illustrative example, consider the case where the base learner for YARDS is the C4.5 decision tree learner, without using subtree raising for pruning. The complexity of the C4.5 build routine is $O(ba \log a) = O(nq \log q)$ [Witten and Frank, 2005], so the overall computational cost of training YARDS with this base learner is $O(n^2 + nq \log q)$.

Theorem 5.5.4. *Let B be a bag, $|B| = m$, and let $L.classify(x) \in O(p(a,b))$, where a is the number of instances used to train L, and b their dimensionality. Then $YARDS.classify(B) \in O(nm + p(q,n))$.*

Proof. As shown earlier, the feature space transformation for B is $O(mn)$. The final step is to apply the base learner L to classify this instance. We showed earlier that the number of instances in F is equal to q, and dimensionality of F is equal to n, so the entire classify procedure is $O(mn + p(q,n))$. □

Continuing the earlier illustrative example, the complexity of the C4.5 decision tree's classification routine is $O(\log a) = O(\log q)$ if we follow Witten and Frank and assume that the decision tree is well balanced. So with C4.5 as the base learner, the complexity of YARDS classification routine is $O(mn + \log q) \in O(mn)$.

Compared to IFLIW, the base learner for YARDS (or MILES) need not be as efficient in the number of instances, since the transformed dataset contains only one instance for each bag. The base learner for IFLIW, on the other hand, must be trained on a dataset consisting of the total collection of all instances in all bags in the training data. However, YARDS' base learner is required to be more efficient with respect to feature space dimensionality, as the dimensionality of the transformed space is equal to the total number of instances in the training data, and will typically be much larger than the dimensionality that IFLIW would have to deal with.

5.5.2 Limitations of the Algorithm

Although YARDS with a linear base classifier learns weighted linear threshold-type concepts, it cannot learn any arbitrary weighted linear threshold concept. As de-

scribed above, it represents the weight function of a dataset as a sum of Gaussian functions, each of which has a peak or trough at an instance from the training data. Even though this representation is quite flexible, this may or may not be a good approximation for the weight function of a given weighted linear threshold concept.

It should also be noted that when non-linear base learners are used, YARDS may not learn weighted linear threshold concepts. If the base learner used for YARDS is not linear, all that can be said about the MI concepts that can be learnt by the algorithm is that the class labels of bags must be in some way determined by these features. We can therefore define the MI assumption used by YARDS as:

> *YARDS Assumption*: There is a set of target points, $K = \{x^1, x^2, \cdots, x^{|K|}\}$. The label of a bag X is in some way related to the set of values $\{s_y(x^1, X), s_y(x^2, X), \cdots, s_y(x^{|K|}, X)\}$. Furthermore, K can be well approximated by the set of instances from the training data.

As in MILES, each feature generated by the YARDS transformation represents a measure of similarity between a bag and a candidate target point. For YARDS, this measure is related to the distance of each instance in the bag from the candidate target point. Because the influence of each instance on a feature is determined by a Gaussian function, instances that are close to the target point will have a far greater influence than those which are further away.

Naturally, YARDS can only learn MI concepts where the YARDS assumption holds. An appropriate base learner must also used — one that is able to learn the relationship between the $s_y(x^k, X)$ values and bag-level class labels for the given MI concept.

5.5.3 Evaluation on Artificial Data

To empirically investigate the efficacy of YARDS, an artificial dataset was created using the WLT1 concept (described in Section 5.4.2) as the generative model. This dataset contained 75 positive and 75 negative bags, with ten instances in each bag. As the instance space was binary rather than numeric, a distance metric for this space was required to compute the similarity functions for MILES and YARDS' feature space transformations. For this, we used the following distance function:

Table 5.11: YARDS vs MILES on $WLT1$: Classification Accuracy

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MILES	46.7	46.7	46.7	46.7	46.7	46.7	46.7	47.9	49.2
YARDS	47.6	100	100	91.2	100	100	46.7	80.2	49.2
Significance	–	∘	∘	∘	∘	∘	–	∘	–

∘, •, – statistically significant improvement, degradation or no difference vs MILES

Table 5.12: YARDS vs MILES on $WLT1$: Root Mean Squared Error

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MILES	0.50	0.50	0.50	0.56	0.50	0.50	0.50	0.50	0.50
YARDS	0.50	0.00	0.00	0.39	0.00	0.00	0.50	0.32	0.50
Significance	–	∘	∘	∘	∘	∘	–	∘	–

∘, •, – statistically significant improvement, degradation or no difference vs MILES

$$d(x_1, x_2) = \begin{cases} 0 & x_1 = x_2 \\ 1 & x_1 \neq x_2 \ . \end{cases}$$

YARDS, MILES and MIWrapper were compared on this dataset via 10×10-fold cross-validation, using the base learners from the MILES experiments described in Chapter 4. The σ parameter for both MILES and YARDS was set to $\sqrt{8 \times 10^5}$, as used by [Chen et al., 2006] for MILES on the *musk2* dataset. Statistical significance was determined using a paired corrected t-test with confidence level $\alpha = 0.05$. The results are summarized in Tables 5.11 — 5.14.

The results show that the performance of YARDS varies dramatically between base learners. Linear classifiers, support vector machines with non-linear RBF kernels and boosted decision stumps performed the best. All of those base learners except for the 1-norm support vector machine achieved perfect classification performance (Table 5.11) and negligible root mean squared error rates (Table 5.12), though the 1-norm SVM still performed competitively with 91.2% accuracy. However, C4.5 was unable to perform better than chance, and neither bagging nor boosting the base learner improved this result.

In contrast to YARDS, MILES' performance was consistently poor regardless of the base learner. MILES performed no better than chance for all base classifiers tried in the experiment (Tables 5.11 and 5.12). For both classification accuracy and the root mean squared error estimator, YARDS enjoyed significant wins over MILES for six of the nine base learners, with no significant losses.

Although YARDS achieved excellent accuracy and root mean squared error results with several base learners, MIWrapper was more consistently strong in terms

Table 5.13: YARDS vs MIWrapper on $WLT1$: Classification Accuracy

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MIWrapper	100	100	100	92.3	100	100	100	100	100
YARDS	47.6	100	100	91.2	100	100	46.7	80.2	49.2
Significance	•	–	–	–	–	–	•	•	•

○, •, – statistically significant improvement, degradation or no difference vs MIWrapper

Table 5.14: YARDS vs MIWrapper on $WLT1$: Root Mean Squared Error

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MIWrapper	0.46	0.46	0.46	0.44	0.46	0.46	0.46	0.46	0.46
YARDS	0.50	0.00	0.00	0.39	0.00	0.00	0.50	0.32	0.50
Significance	•	○	○	○	○	○	•	○	•

○, •, – statistically significant improvement, degradation or no difference vs MIWrapper

of classification accuracy. The MIWrapper method achieved one-hundred percent accuracy with eight of the nine base learners (Table 5.13). The worst performing base learner for MIWrapper was the 1-norm support vector machine, where an accuracy of 92.3% was observed. This result was statistically indistinguishable to the accuracy of YARDS using the same base classifier.

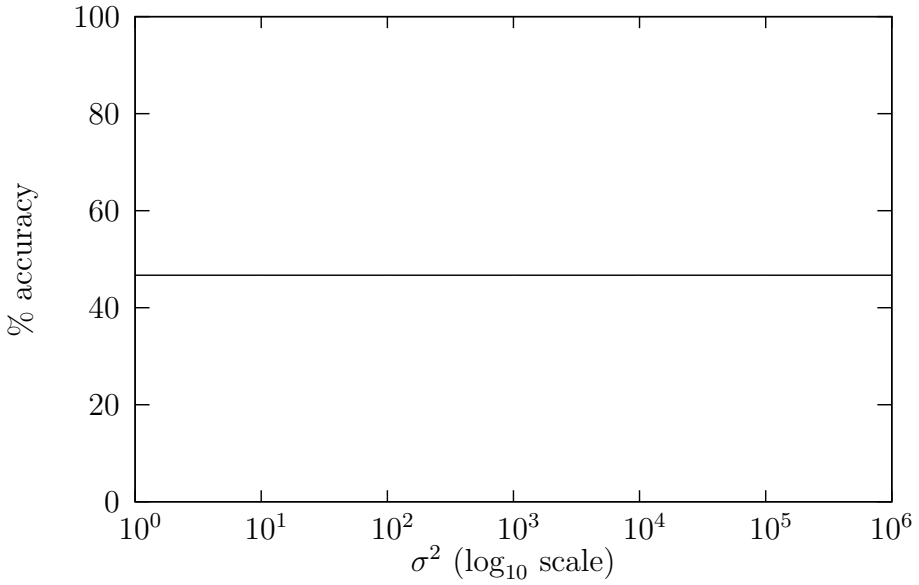
However, YARDS fared better than MIWrapper overall with respect to the root mean squared error rate, with six significant wins and three losses for that performance measure (Table 5.14). MIWrapper consistently produced a root mean squared error rate of approximately 0.46, and never approached the negligible RMSE values enjoyed by YARDS using four of the nine base learners.

Further Investigation — Parameter Tuning

The above experimental results motivated two questions: why did MILES perform so poorly compared to the other two methods, and why was there such a dramatic variability between the performance of base learners for YARDS? We considered the possibility that the selection of the σ parameter for the MILES and YARDS transformations may have been a factor in this result.

To investigate the first question, MILES was evaluated on the WLT1 dataset with a range of different σ values. The support vector machine with the linear kernel was selected as the base learner for the experiment, as that algorithm had performed strongly as a base classifier for both YARDS and MIWrapper. The σ parameter was varied between $\sigma^2 = 1$ and $\sigma^2 = 10^6$, using the step procedure $\sigma^2_{i+1} = \sigma^2_i \times 10$, and the algorithm was evaluated using a single ten-fold cross-validation. It was found that the alteration of the σ value made no difference to the classification performance

Figure 5.11: Parameter Tuning — MILES with linear kernel SVM on $WLT1$



of the algorithm. The results are shown in Figure 5.11.

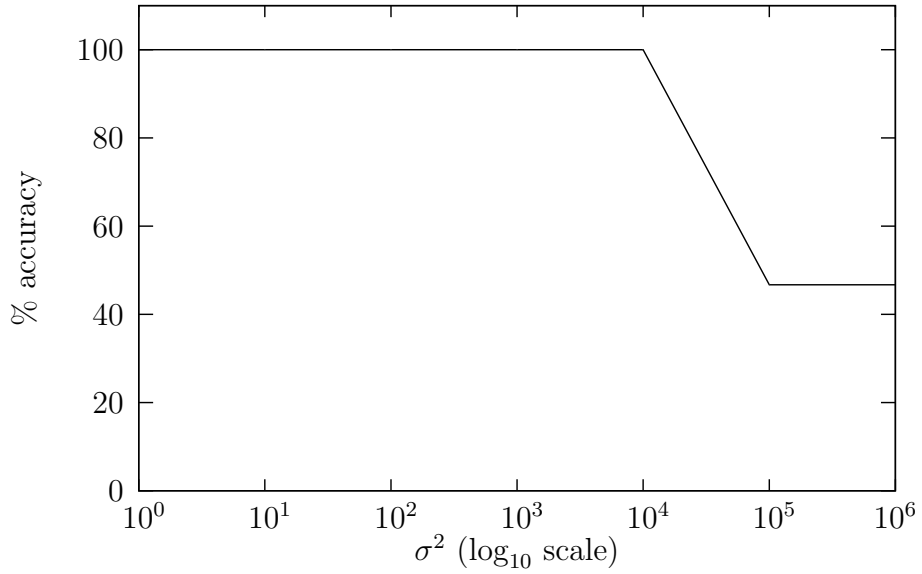
Combined with the results described in the previous section, this result indicates that MILES may not be able to learn the $WLT1$ concept, regardless of the base learner used or the value of the σ parameter. An explanation for this from a theoretical point of view is provided in the next section.

A similar experiment was performed to determine whether the poor performance of YARDS with C4.5 as the base learner was due to the value of the σ parameter or a more fundamental reason. YARDS with C4.5 was evaluated on the dataset using the same evaluation method and parameter values as for the MILES tuning experiment. The results of the experiment are provided in Figure 5.12.

It was found that YARDS with C4.5 achieved one-hundred percent classification accuracy for all parameter values where σ was less than or equal to 10^4, thus demonstrating that YARDS with C4.5 was capable of learning this type of concept when appropriate parameter values were selected. To verify that this was not merely due to a chance effect caused by the particular dataset that was generated, an alternative test dataset was created using the $WLT1$ generative model. This dataset was created in the same way as the original $WLT1$ data, but using a different random seed. It also contained 75 positive and 75 negative bags, with each bag consisting of ten instances.

All experiments were repeated on this alternative dataset using the parameter value $\sigma^2 = 10^4$ for both YARDS and MILES. With this new σ value, YARDS

Figure 5.12: Parameter Tuning — YARDS with C4.5 on $WLT1$



achieved one-hundred percent classification accuracy with all base learners tried in the experiment. YARDS also had negligible root mean squared error values for all base learners except for the 1-norm SVM which had an RMSE of 0.18. MILES did not significantly improve with the new σ value, with an accuracy rate of less than fifty percent for all base learners. The results are shown in Tables 5.15 — 5.18.

An Explanation for MILES' Poor Performance on $WLT1$

As shown earlier, the $WLT1$ concept can be written in the form:

$$\nu_{WLT1}(X) = \text{sign}\Big(2 \times count(X_i = t) - count(X_i = f) - 3\Big) \ ,$$

where $count(X_i = x)$ is the number of instances X_i in the bag X where the attribute value for X_i is equal to x.

Written this way, it is clear that the computation of $WLT1(X)$ is difficult to achieve if the learning program cannot represent the number of occurrences of the target points t and f in a bag. MILES is unable to represent this type of concept because the feature space mapping does not include any information about the number of occurrences of instances close to (or at) a target point, but only the distance from that point to the closest instance in the bag. The "max" operator used in the MILES transformation selects the closest instance to a target point, and ignores all other instances that may be close to or at that point. Because of this, the

Table 5.15: YARDS vs MILES on $WLT1$ Alternative Dataset, $\sigma^2 = 10^4$: Classification Accuracy

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MILES	46.7	46.7	46.7	46.7	46.7	46.7	46.7	47.9	49.2
YARDS	100	100	100	100	100	100	100	100	100
Significance	∘	∘	∘	∘	∘	∘	∘	∘	∘

∘, •, − statistically significant improvement, degradation or no difference vs MILES

Table 5.16: YARDS vs MILES on $WLT1$ Alternative Dataset, $\sigma^2 = 10^4$: Root Mean Squared Error

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MILES	0.50	0.50	0.50	0.56	0.50	0.50	0.50	0.50	0.50
YARDS	0.00	0.00	0.00	0.18	0.00	0.00	0.00	0.00	0.00
Significance	∘	∘	∘	∘	∘	∘	∘	∘	∘

∘, •, − statistically significant improvement, degradation or no difference vs MILES

Table 5.17: YARDS vs MIWrapper on $WLT1$ Alternative Dataset, $\sigma^2 = 10^4$: Classification Accuracy

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MIWrapper	100	100	100	93.9	100	100	100	100	98.9
YARDS	100	100	100	100	100	100	100	100	100
Significance	−	−	−	∘	−	−	−	−	−

∘, •, − statistically significant improvement, degradation or no difference vs MIWrapper

Table 5.18: YARDS vs MIWrapper on $WLT1$ Alternative Dataset, $\sigma^2 = 10^4$: Root Mean Squared Error

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
MIWrapper	0.46	0.46	0.46	0.44	0.46	0.46	0.46	0.46	0.46
YARDS	0.00	0.00	0.00	0.18	0.00	0.00	0.00	0.00	0.00
Significance	∘	∘	∘	∘	∘	∘	∘	∘	∘

∘, •, − statistically significant improvement, degradation or no difference vs MIWrapper

features generated by the MILES transformation are related to the distance from each target point to the closest instance in the bag, but do not contain any further information pertaining to the number of instances that are close to each target point.

In the case of $WLT1$, the target points are t and f. The features created by the MILES transformation describe the closest points to t and f, which in this problem corresponds to the existence or non-existence of the target points in the bag. In contrast, a feature generated by the YARDS transformation represents the sum of values computed based on the distance between each instance in the bag and a target point. For $WLT1$, a t instance will add a larger value to the attribute corresponding to t than to the instance corresponding to f, and vice-versa. Thus, the feature values are directly related to the number of occurrences of t and f, and therefore YARDS gives the base learner adequate information from which to learn the $WLT1$ concept, while MILES does not.

Experiments on Weighted Collective Assumption Artificial Problems

We showed in Section 5.4.3 that any deterministic weighted collective assumption can be represented as a weighted linear threshold assumption. The non-deterministic case can be interpreted as adding class noise that is caused by the probabilistic labeling process. Because of this relationship between the assumptions, we investigated the performance of YARDS on datasets generated using the weighted collective MI assumption.

YARDS was evaluated on the 64 artificial weighted collective assumption datasets described in Section 5.3.3, using the base learners from the MILES experiment (Chapter 4). The experiments were performed using both MIWrapper and MILES as the baseline algorithms for comparison. For both MILES and YARDS, σ^2 was set to 8×10^5, as used by [Chen et al., 2006] for MILES on the *musk2* dataset. Ten repeats of 10-fold cross validation were performed on each dataset using each base learner, and statistical significance was determined using a pairwise resampled corrected t-test with significance level $\alpha = 0.05$.

Just as in the experiments described in Section 5.3.3, it was found that the random forests base learner was creating unreasonably large trees on many of the datasets. This was a problem for all three MI algorithms, but was particularly severe for YARDS and MILES, where execution slowed to a crawl. To resolve this problem,

Table 5.19: YARDS: Significant Wins and Losses vs MIWrapper on Weighted Collective Artificial Data (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	2	8	12	6	11	3	2	4	0
Losses	56	16	18	27	0	20	56	9	55
Ties	6	40	34	31	53	41	6	51	9
Wins - Losses	-54	-8	-6	-21	11	-17	-54	-5	-55
Average MIWrapper	75.1	68.0	68	67.1	68.0	77.9	76.2	64.5	77.9
Average YARDS	49.6	64.2	63.3	59.7	69.5	71.4	49.6	62.0	52.1

Table 5.20: YARDS: Significant Wins and Losses vs MIWrapper on Weighted Collective Artificial Data (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	0	17	15	14	17	15	0	8	0
Losses	53	12	17	38	12	17	53	16	57
Ties	11	35	32	12	35	32	11	40	7
Wins - Losses	-53	5	-2	-24	5	-2	-53	-8	-57
Average MIWrapper	0.43	0.46	0.46	0.47	0.46	0.43	0.43	0.47	0.43
Average YARDS	0.50	0.44	0.44	0.50	0.42	0.41	0.50	0.47	0.50

the random forests base learner was limited to a maximum depth of 20 levels per tree. For this reason the results reported for MIWrapper using this base learner may differ from those in Section 5.3.3.

The results are summarized in Tables 5.19 — 5.22. For all base learners except logistic regression and the 2-norm support vector machine with the linear kernel, YARDS exhibited a larger number of significant losses than significant wins against MIWrapper for both classification accuracy (Table 5.19) and the root mean squared error (Table 5.20). With logistic regression as the base learner, YARDS achieved 11 wins with no losses against MIWrapper for classification accuracy, but its average accuracy was only 1.5 percentage points higher than MIWrapper. Although YARDS had five more wins than losses against MIWrapper with the root mean squared error measure when the 2-norm SVM with the linear kernel was used as the base learner, it also exhibited eight more losses than wins under the accuracy performance measure.

Adaboost with decision stumps was the best performing base learner for all three MI algorithms. Using this base learner, the average accuracies were 77.9%, 76.9% and 71.4% for MIWrapper, MILES and YARDS, respectively. The lowest root mean squared error rate for each MI algorithm was also achieved using this base learner.

The worst-performing base learners for YARDS were C4.5 and its bagged and

Table 5.21: YARDS: Significant Wins and Losses vs MILES on Weighted Collective Artificial Data (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	13	4	6	33	9	2	13	43	10
Losses	0	32	27	0	9	18	0	0	0
Ties	51	28	31	31	46	44	51	21	54
Wins - Losses	13	-24	-21	33	0	-16	13	43	10
Average MILES	46.7	74.4	72.7	46.7	69.7	76.9	46.7	48.9	49.3
Average YARDS	49.6	64.2	63.3	59.7	69.5	71.4	49.6	62.0	52.1

Table 5.22: YARDS: Significant Wins and Losses vs MILES on Weighted Collective Artificial Data (Root Mean Squared Error)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	1	3	7	38	41	7	2	29	3
Losses	0	27	25	16	12	15	0	1	0
Ties	63	34	32	10	11	42	62	34	61
Wins - Losses	1	-24	-18	22	29	-8	2	28	3
Average MILES	0.50	0.37	0.39	0.56	0.50	0.36	0.50	0.50	0.50
Average YARDS	0.50	0.44	0.44	0.50	0.42	0.41	0.50	0.47	0.50

boosted variants, each of which achieved an average accuracy of only around fifty percent (Table 5.19). This figure is slightly misleading, as these base learners were able to achieve accuracies higher than fifty percent on 13 (in the case of C4.5 and Adaboost with C4.5) and 25 (in the case of bagged C4.5) of the 64 datasets. The algorithm had an accuracy level of 46.7% on all of the other datasets (except for bagged C4.5, where the value was typically 49.27%, and occasionally as low as 43.53%), which brought the average down to around fifty percent.

The YARDS method performed more favourably compared to MILES than to MIWrapper. The relative performance of YARDS and MILES depended to a surprisingly large degree on the base classifier used. With five of the nine base learners, YARDS achieved a higher number of significant wins than losses against MILES for both accuracy (Table 5.21) and the root mean squared error value (Table 5.22). Considering all of the base learners together, YARDS was ahead overall with a total of 51 more significant wins than losses for classification accuracy, and 35 more wins than losses for the root mean squared error measure.

In the case of each of the five base learners where YARDS achieved a higher number of wins than losses, the average accuracy of MILES was slightly less than fifty percent. Except for the 1-norm support vector machine, all of these base learners were decision tree learners. MILES with C4.5, the 1-norm support vector machine,

and with boosted C4.5 each had an accuracy rate of 46.7% on all 64 datasets, while MILES with bagged C4.5 had an accuracy of 49.3% on all datasets. MILES with random forests had classification accuracy rates ranging between 47.9% and 53.7%. Although the classification accuracy of YARDS with C4.5, and with bagged and boosted C4.5, was equal to MILES' poor performance on most datasets, the algorithm performed better in some cases, resulting in a number of significant wins (and no losses) for those base learners despite poor overall average accuracy rates.

Though YARDS performed better than MILES for the majority of the base learners, the best MILES classifiers performed better (in both accuracy and root mean squared error rate) than the best performing YARDS classifiers. The top three base learners for MILES were Adaboost with decision stumps, and the 2-norm SVM with both the linear and the polynomial kernel, achieving average accuracies of 76.9%, 74.4% and 72.7% (Table 5.21), and root mean squared error values of 0.36, 0.37 and 0.39, respectively (Table 5.22). For YARDS, the top three base learners for classification accuracy were Adaboost with decision stumps, logistic regression and the 2-norm SVM with the linear kernel, with average accuracy rates of 71.4%, 69.5%, and 64.2%, respectively. These base learners were also the best for the root mean squared error measure, with RMSE rates of 0.41, 0.42, and 0.44 respectively, though the support vector machine with the RBF kernel also had an RMSE rate of 0.44.

YARDS and MILES tied in the number of significant wins and losses for classification accuracy using the logistic regression base learner, and were also very similar in terms of average accuracy. Despite this, YARDS with logistic regression had 41 significant wins with only 12 losses for the root mean squared error performance measure. Thirty of these significant wins, and no losses, were achieved on datasets generated by a non-deterministic classification process, with the other 11 wins and 12 losses on deterministically labeled datasets (not shown in the tables).

Except for the case of logistic regression, if YARDS or MIWrapper were ahead in terms of significant wins minus significant losses for classification accuracy, they were also ahead for the root mean squared error rate, although the magnitude of the difference between significant wins and losses was generally smaller for the RMSE performance measure than for accuracy. Because these performance measures were thus connected in the observed results, the root mean squared error measure is not

Table 5.23: YARDS: Significant Wins and Losses vs MILES on Weighted Collective Artificial Data with Deterministic Generative Model (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	8	4	5	26	6	2	8	30	7
Losses	0	22	18	0	9	15	0	0	0
Ties	24	6	9	6	17	15	24	2	25
Wins - Losses	8	-18	-13	26	-3	-17	8	30	7
Average MILES	46.7	87.7	84.3	46.7	83.2	92.6	46.7	49.0	49.3
Average YARDS	50.6	72.3	70.4	69.1	79.0	83.7	50.6	69.1	53.4

Table 5.24: YARDS: Significant Wins and Losses vs MILES on Weighted Collective Artificial Data with Non-Deterministic Generative Model (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	5	0	1	7	3	0	5	13	3
Losses	0	10	9	0	0	3	0	0	0
Ties	27	22	22	25	29	29	27	19	29
Wins - Losses	5	-10	-8	7	3	-3	5	13	3
Average MILES	46.7	61.2	61.1	46.7	56.2	61.1	46.7	48.5	49.3
Average YARDS	46.6	56.1	56.1	50.3	50.0	59.2	48.6	55.0	50.8

considered any further in following, where we investigate classification determinism and bag size.

As expected, both YARDS and MILES performed better on datasets created with a deterministic generative model than on datasets where a probabilistic labeling process was used (Tables 5.24 and 5.24). With boosted decision stumps as the base learner, MILES achieved an impressive accuracy rate of 92.6% on the deterministically labeled datasets, while YARDS' accuracy rate with that base learner was 83.7%. There was no clear trend regarding the performance of these algorithms relative to each other between deterministically and probabilistically labeled datasets — the number of significant wins minus losses for YARDS was higher on deterministically labeled datasets for five of the nine base learners, and higher on probabilistically labeled datasets for the other four single-instance learners.

Consistent with the results of the experiments using IFLIW and MIWrapper on these artificial problems, MILES and YARDS performed better (or at least equally well) on the datasets with random bag sizes (Table 5.26) than the datasets with fixed bag sizes (Table 5.25), presumably because the randomly sized bags were smaller on average than the fixed size bags. While bag-size determinism did not in general make a clear difference in the relative performance of MILES and YARDS, the significant wins enjoyed by YARDS over MILES for the C4.5 (plain, bagged, and boosted) base

Table 5.25: YARDS: Significant Wins and Losses vs MILES on Weighted Collective Artificial Data with Fixed Bag Sizes (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	0	4	6	12	9	1	0	17	0
Losses	0	9	5	0	4	9	0	0	0
Ties	32	19	21	19	19	22	32	15	32
Wins - Losses	0	-5	1	12	5	-8	0	17	0
Average MILES	46.7	73.7	72.5	46.7	68.2	75.2	46.7	48.1	49.3
Average YARDS	46.7	70.2	71.6	61.3	71.0	68.8	46.7	57.2	49.3

Table 5.26: YARDS: Significant Wins and Losses vs MILES on Weighted Collective Artificial Data with Random Bag Sizes (Classification Accuracy)

	C4.5	SMO (Lin)	SMO (RBF)	1-Norm SVM	Logistic	Adaboost + D. Stump	Adaboost + C4.5	Random Forest	Bagging + C4.5
Wins	13	0	0	21	0	1	13	26	10
Losses	0	23	22	0	5	9	0	0	0
Ties	19	9	10	11	27	22	19	6	22
Wins - Losses	13	-23	-22	21	-5	-8	13	26	10
Average MILES	46.7	75.1	72.9	46.7	71.2	78.5	46.7	49.4	49.3
Average YARDS	52.6	58.1	54.9	58.6	68.0	74.1	52.5	66.9	55.0

learners were all achieved on datasets with random bag sizes.

5.5.4 Conclusions

It was found that YARDS was able to achieve perfect classification accuracy and a root mean squared error rate of zero for the $WLT1$ data when an appropriate value for the σ parameter was used, for all base learners tried in the experiment. This is in contrast to MILES, which was unable to achieve an accuracy level better than chance, regardless of the base learner or σ value. We showed that MILES is unable to represent "counting" concepts such as $WLT1$, thereby explaining its failure to learn in this domain.

The MIWrapper algorithm was also able to achieve perfect classification accuracy for $WLT1$ with all base learners except for the 1-norm SVM, but YARDS was superior for the root mean squared error performance measure. An advantage of MIWrapper over YARDS was that no parameter tuning was required.

However, even though all deterministic weighted collective MI concepts can be represented as weighted linear threshold concepts (and from a classification point of view, probabilistic weighted collective concepts are deterministic concepts with class noise), YARDS was not able to perform as well as MIWrapper on the set of 64 artificial weighted collective concepts. In contrast, YARDS was very competitive

with MILES on this data with a larger number of significant wins than losses on five of the nine datasets, and with more significant wins than losses against MILES overall. Adaboost with decision stumps was the best base learner for all three MI algorithms, with 77.9%, 76.9% and 71.4% accuracy on average for MIWrapper, MILES and YARDS respectively.

It should be noted that YARDS is only guaranteed to learn weighted linear threshold concepts (and by extension, weighted collective concepts) when the base learner is linear. Even when this condition is satisfied, YARDS can only model weighted linear threshold concepts if the weight function can be approximated by a sum of a set of Gaussian functions centred on points from the training data. These reasons may partially explain YARDS' mediocre performance on these datasets. A lack of parameter tuning for YARDS and MILES may also have been a factor in these results.

Chapter 6
Evaluation of the New Algorithms on Real-World Data

This chapter presents an empirical investigation into the predictive performance of the new algorithms introduced in Chapter 5 on real-world data. The IFLIW and YARDS algorithms are evaluated on a diverse array of real-world datasets using a wide variety of base learners, with comparison to the related approaches MIWrapper and MILES.

6.1 Experiment Design

The algorithms were evaluated with the same datasets and base learners that were used for the MILES experiments (described in Chapter 4). The number of iterations μ for IFLIW was set to 20, and the regression base learner was additive regression with 100 decision stumps. For both MILES and YARDS, the parameter value $\sigma^2 = 8 \times 10^5$ was used, as selected by [Chen et al., 2006] for MILES on the *musk2* data. The evaluation method was 10×10-fold cross-validation. A paired corrected t-test with significance level $\alpha = 0.05$ was used to determine significant differences between the algorithms. Performance was measured using classification accuracy and the root mean squared error.

MIWrapper was used as a baseline algorithm for comparison in the experiments. It was found by [Dong, 2006] and confirmed in the experiments in Chapter 4 that despite its simplicity, MIWrapper is very competitive with the more sophisticated state-of-the-art MI learning methods, and is therefore a good benchmark for evaluating MI algorithms. Results for the MILES algorithm are also provided for comparison. As described in Chapter 5, IFLIW is closely related to the MIWrapper algorithm, and YARDS is closely related to MILES.

Table 6.1: Average Performance Over All Datasets (Classification Accuracy)

Base learner	MIWRAPPER	MILES	IFLIW	YARDS
C4.5	73.8	70.8	72.5	68.8
Random Forest	77.0	78.5	77.6	73.0
Adaboost + C4.5	76.2	73.3	75.0	70.1
Adaboost + D. Stump	75.4	79.5	72.7	72.7
Bagging + C4.5	76.4	73.6	74.5	72.1
SMO (LIN)	70.2	74.2	71.1	69.3
SMO (RBF)	70.9	73.2	72.5*	68.8
1-Norm SVM	70.8	73.0	71.0	75.2
Logistic	72.9	74.6	70.4	74.47

*Based on incomplete results for *thioredoxin* (26/100 folds)

Table 6.2: Average Performance Over All Datasets (Root Mean Squared Error)

Base learner	MIWRAPPER	MILES	IFLIW	YARDS
C4.5	0.41	0.47	0.42	0.48
Random Forest	0.39	0.38	0.38	0.42
Adaboost + C4.5	0.39	0.44	0.39	0.47
Adaboost + D. Stump	0.41	0.37	0.41	0.42
Bagging + C4.5	0.39	0.40	0.41	0.42
SMO (LIN)	0.42	0.42	0.43	0.44
SMO (RBF)	0.42	0.47	0.42*	0.43
1-Norm SVM	0.43	0.42	0.44	0.40
Logistic	0.44	0.45	0.44	0.41

*Based on incomplete results for *thioredoxin* (26/100 folds)

6.2 Experimental Results and Analysis

In this section, the experimental results are presented in summary form, and the implications of these results are discussed. The results are presented in more detail in Appendix 7.2.

6.2.1 Average Performance Over All Datasets

The average (arithmetic mean) performance of each algorithm over all nine datasets is shown in Tables 6.1 and 6.2. It should be noted that the datasets were not all of equal learning difficulty, and were diverse in the level of class skewness, number of bags and number of instances. Small datasets were given equal consideration to larger ones when computing the averages. These factors should be considered when interpreting the results for this section.

In the observed results, the average classification accuracy (Table 6.1) and root mean squared error rate (Table 6.2) for IFLIW were typically very similar to the results for MIWrapper using the same base learner. In terms of average performance, the random forests classifier was the best base learner for both IFLIW and MIWrapper under both performance measures. Using that base classifier, MIWrapper had an average accuracy of 77.0% with a root mean squared error of 0.39, while

IFLIW's average accuracy was 77.6% with an RMSE of 0.38. The boosted decision stumps classifier was the best performing base learner for MILES in terms of average performance, with an accuracy rate of 79.5% and an RMSE of 0.37.

For each base learner except the 1-norm support vector machine and logistic regression, YARDS performed slightly worse than the other three schemes, though the differences were not very large. With the 1-norm SVM base learner, YARDS was slightly ahead of the other algorithms for both performance metrics. YARDS also had the lowest average root mean squared error rate of the four schemes when the logistic regression base learner was used. The 1-norm SVM base learner was the best performer on average for YARDS, with a classification accuracy of 75.2% and a root mean squared error of 0.40.

6.2.2 Significant Wins and Losses vs MIWrapper

Tables 6.3 and 6.4 show the significant differences in performance versus the MIWrapper algorithm for each of the other three schemes. For each scheme, the first column shows the number of significant wins, the second column displays the number of significant losses, and the final column shows the number of significant wins minus significant losses against MIWrapper. Once again, the different sizes of the datasets should be taken into consideration when interpreting this result — significant differences were much harder to obtain for datasets with few bags.

It was found that all three algorithms exhibited less wins than losses against MIWrapper in total over all of the base learners, for both accuracy (Table 6.3) and root mean squared error (Table 6.4). MILES and IFLIW both exhibited a greater number of significant losses for the RMSE measure than for classification accuracy, while the number of significant wins was similar between the two metrics. YARDS had a similar number of wins and losses for both metrics.

For accuracy and root mean squared error, the number of significant wins and significant losses for MILES were each greater than for IFLIW, but the two algorithms had similar overall total wins minus losses. YARDS was the worst of the three schemes in terms of total wins minus losses, with 34 more losses than wins for both performance metrics. A closer inspection of the results provided in Appendix 7.2 shows that the majority of YARDS' significant losses against MIWrapper were

Table 6.3: Significant Differences vs MIWrapper Over All Datasets: Classification Accuracy (Significant Wins / Losses / Wins - Losses)

Base learner	MILES			IFLIW			YARDS			Total		
C4.5	0	3	-3	0	0	0	0	7	-7	0	10	-10
Random Forest	0	3	-3	0	2	-2	0	7	-7	0	12	-12
Adaboost + C4.5	0	2	-2	0	0	0	0	7	-7	0	9	-9
Adaboost + D. Stump	3	1	2	1	0	1	2	5	-3	6	6	0
Bagging + C4.5	0	3	-3	0	4	-4	0	6	-6	0	13	-13
SMO (LIN)	3	2	1	1	1	0	2	4	-2	6	7	-1
SMO (RBF)	4	2	2	1	1	0	2	4	-2	7	7	0
1-Norm SVM	3	3	0	0	1	-1	2	4	-2	5	7	-2
Logistic	2	3	-1	1	1	0	2	0	2	5	4	1
Total	15	22	-7	4	10	-6	10	44	-34			

Table 6.4: Significant Differences vs MIWrapper Over All Datasets: Root Mean Squared Error (Significant Wins / Losses / Wins - Losses)

Base learner	MILES			IFLIW			YARDS			Total		
C4.5	0	6	-6	0	1	-1	0	7	-7	0	14	-14
Random Forest	2	3	-1	0	2	-2	0	7	-7	2	12	-10
Adaboost + C4.5	0	6	-6	1	1	0	0	7	-7	1	14	-13
Adaboost + D. Stump	3	3	0	1	1	0	0	5	-5	4	9	-5
Bagging + C4.5	0	3	-3	0	6	-6	0	7	-7	0	16	-16
SMO (LIN)	2	2	0	1	2	-1	2	5	-3	5	9	-4
SMO (RBF)	0	7	-7	1	2	-1	2	3	-1	3	12	-9
1-Norm SVM	5	3	2	1	5	-4	4	2	2	10	10	0
Logistic	1	6	-5	0	5	-5	3	2	1	4	13	-9
Total	13	39	-26	5	25	-20	11	45	-34			

on image problems, while all but one of its significant wins were on the *mutagenesis* datasets.

None of the three algorithms exhibited any significant wins against MIWrapper for classification accuracy using the C4.5 (plain, bagged and boosted) and random forests base learners. Those four base learners were also the worst for YARDS in terms of significant differences against MIWrapper, with six or seven losses for each.

6.2.3 The Best Results for Each Scheme

The best results for each scheme on each dataset are displayed in Table 6.5 (Classification Accuracy) and Table 6.6 (root mean squared error). The top-performing base learners for each scheme on each dataset were typically very similar to each other under both peformance measures, although there were several exceptions to this.

Notable exceptions were MILES' best results for *musk2* and *eastwest* / *westeast*, which were 8 – 12 percentage points ahead of each of the other algorithms, and were also noticeably superior in the root mean squared error rate. With the 1-norm SVM, MILES achieved 90.4% accuracy on *musk2*, while the other three algorithms had accuracy rates of around 82.8%. It should be noted that the σ parameter for MILES

Table 6.5: The Best Result For Each Scheme (Classification Accuracy)

Dataset	Best MIWrapper	%	Best MILES	%	Best IFLIW	%	Best YARDS	%
musk1	Random Forest	87.3	Adaboost + D. Stump	88.0	Adaboost + C4.5	86.8	Random Forest	84.4
musk2	SMO (RBF)	82.8	1-Norm SVM	90.4	SMO (LIN)	82.8	1-Norm SVM	82.6
eastwest	Adaboost + D. Stump	69.0	Adaboost + D. Stump	81.0	Random Forest	71.0	Random Forest	72.5
westeast	Adaboost + D. Stump	69.0	Adaboost + D. Stump	81.0	Random Forest	69.0	Random Forest	72.5
mutagenesis-atoms	Random Forest	81.9	Adaboost + D. Stump	83.9	Random Forest	82.0	Random Forest	81.9
mutagenesis-bonds	Random Forest	83.1	Adaboost + D. Stump	86.3	C4.5	83.8	Random Forest	82.6
mutagenesis-chains	Bagging + C4.5	85.3	Adaboost + D. Stump	86.0	C4.5	84.1	Logistic	83.9
suramin	1-Norm* SVM	65.0	1-Norm* SVM	65.0	SMO (RBF)	93.5	1-Norm SVM	65.0
thioredoxin	Adaboost + C4.5	88.0	Adaboost + D. Stump	89.3	Adaboost + C4.5	88.5	Bagging + C4.5	89.7
elephant	Random Forest	87.1	1-Norm SVM	84.3	Adaboost + C4.5	84.4	Logistic	83.9
fox	Adaboost + D. Stump	65.7	Random Forest	64.9	C4.5*	64.1	Logistic	55.9
tiger	Random Forest	84.3	1-Norm SVM	82.0	Random Forest	81.3	1-Norm SVM	81.3
bikes	Bagging + C4.5	82.5	SMO (LIN)	80.5	SMO (LIN)	81.6	1-Norm SVM	83.9
cars	Random Forest	74.8	SMO (LIN)	72.5	Random Forest	72.4	Logistic	72.2
people	Random Forest	82.6	Random Forest	77.5	Adaboost + C4.5	81.1	1-Norm SVM	78.5

∗ Scheme was best-equal with one or more other schemes.

Table 6.6: The Best Result For Each Scheme (Root Mean Squared Error)

Dataset	Best MIWrapper	%	Best MILES	%	Best IFLIW	%	Best YARDS	%
musk1	Bagging + C4.5	0.29	Adaboost + D. Stump	0.26	Adaboost + C4.5	0.30	Random Forest	0.33
musk2	SMO (RBF)	0.33	SMO (LIN)	0.27	SMO (LIN)	0.34	1-Norm SVM	0.38
eastwest	SMO (RBF)	0.46	Adaboost + D. Stump	0.27	Random Forest	0.39	SMO (RBF)	0.39
westeast	SMO (RBF)	0.46	Adaboost + D. Stump	0.27	Random Forest	0.39	SMO (RBF)	0.39
mutagenesis-atoms	Random* Forest	0.36	Adaboost + D. Stump	0.35	Random Forest	0.36	Random Forest	0.39
mutagenesis-bonds	Random* Forest	0.36	Adaboost + D. Stump	0.33	C4.5*	0.35	Random Forest	0.36
mutagenesis-chains	Bagging + C4.5	0.33	Adaboost + D. Stump	0.32	C4.5*	0.34	Logistic	0.34
suramin	1-Norm SVM	0.44	SMO (RBF)	0.37	Adaboost + C4.5	0.34	1-Norm SVM	0.44
thioredoxin	Random Forest	0.30	Random Forest	0.30	Adaboost* + D. Stump	0.31	Bagging + C4.5	0.29
elephant	Adaboost* + C4.5	0.33	SMO (LIN)	0.34	Adaboost + C4.5	0.33	Logistic	0.33
fox	Bagging + C4.5	0.46	Random Forest	0.47	Random Forest	0.47	SMO (LIN)	0.50
tiger	Adaboost* + D. Stump	0.36	SMO (LIN)	0.36	Random Forest	0.36	1-Norm* SVM	0.38
bikes	Adaboost + C4.5	0.36	1-Norm SVM	0.40	Adaboost + C4.5	0.36	1-Norm SVM	0.35
cars	Adaboost + C4.5	0.43	Random Forest	0.43	Adaboost + C4.5	0.43	1-Norm* SVM	0.44
people	Bagging* + C4.5	0.37	Random Forest	0.39	Adaboost + C4.5	0.37	1-Norm SVM	0.39

∗ Scheme was best-equal with one or more other schemes.

was selected based on the results of tuning experiments by [Chen et al., 2006] on a subset of this data, so this result may be slightly optimistic. MILES with boosted decision stumps achieved an accuracy of 81.0% on both *eastwest* and *westeast*, while the others ranged between accuracies of 69.0–72.5% on those datasets.

Another notable result was that using the 2-norm support vector machine with the radial basis function kernel, IFLIW exhibited an accuracy of 93.5% on the *suramin* dataset. The other algorithms only achieved an accuracy of 65% on this dataset, as did the other base learners for IFLIW. Furthermore, all algorithms tried in the experiments from Chapter 4 also achieved at most 65% on that data, except for SimpleMI (propositionalization with summary statistics) which exhibited an accuracy of 73.0% using the 2-norm support vector machine with the linear kernel. The best result for IFLIW also exhibited a lower root mean squared error value on this dataset than the other MI algorithms, although the difference was not as notable as for classification accuracy. Note that the *suramin* dataset only contains 11 bags, and the result was not statistically significantly superior to the results for the other algorithms.

Although the results for IFLIW on this problem were notable, their incongruity with all of the other results on that dataset made them appear slightly suspicious. A separate experiment was performed to confirm that this was not just a statistical anomaly. IFLIW and MIWrapper using the SVM with the RBF kernel were evaluated on the *suramin* dataset using ten repeats of randomized 90%/10% train/test splits, and the results were averaged over the repeats. In this experiment, the average classification accuracy for IFLIW was 88.9% with a standard deviation of 22.1%, and the accuracy for MIWrapper was 44.4% with a standard deviation of 46.4%.

It was interesting that the best base learners for YARDS were generally very competitive with the best base learners for the other schemes, despite typically lagging slightly behind the other algorithms in terms of average performance and significant differences to MIWrapper. The only major exception to this was the result for the *fox* dataset under the accuracy performance measure, where the best result for YARDS was 55.9% with the logistic regression base learner. The best results for the other algorithms ranged from 64.1 to 65.7 on this dataset. The difference was not as notable for the best results in terms of the root mean squared error measure — YARDS with the 2-norm linear SVM exhibited an RMSE of 0.50,

while the other algorithms' best RMSE values were between 0.46 and 0.47.

6.3 Conclusions for this Study

The experimental results show that IFLIW and YARDS are not generally superior to their predecessors MIWrapper and MILES on the the datasets tried in the study, although they are still very competitive overall.

On average, IFLIW was very similar to MIWrapper and MILES in terms of classification accuracy and root mean squared error (Tables 6.1 and 6.2). Though IFLIW often achieved superior root mean squared error rates to MIWrapper on artificial data, such improved performance was not generally observed on the real-world datasets used in this study.

However, IFLIW achieved an accuracy of 93.5% on the *suramin* dataset using the SVM with the RBF kernel, a result which was nearly 30 percentage points higher than the best results for the other algorithms. This difference was not statistically significant, however, possibly due to the small size of the dataset.

YARDS was generally slightly worse than the other three algorithms for each base learner in terms of both average performance (Tables 6.1 and 6.2) and significant differences to MIWrapper (Tables 6.3 and 6.4). This may partially be due to a lack of parameter tuning performed for YARDS. As parameter tuning for all algorithms and datasets was infeasible, the σ parameter was selected due to known results for MILES on the *musk2* data [Chen et al., 2006]. That parameter value may not have been an optimal value for YARDS.

Even though YARDS was often slightly worse than the other algorithms for a given base learner, the results for the best YARDS base learner on each dataset were generally very similar to the best results for the other three algorithms. This was true both for classification accuracy (Table 6.5) and the root mean squared error measure (Table 6.6). The only notable exception was for the *fox* dataset, where the best YARDS classifier exhibited an accuracy level of 55.9%, while the other algorithms achieved accuracies between 64.1% and 65.7%.

Chapter 7

Conclusions and Future Work

This thesis investigated assumptions and algorithms for learning generalized multiple-instance concepts where instance weights can be used to model the level of contribution that each instance in a bag has on bag-level class labels.

The MILES algorithm [Chen et al., 2006] was identified as a weight-learning MI algorithm as it assigns a weight to each instance in the training bags, and further learns a weight function over the Cartesian product of instance space and bag space. An empirical study was performed to investigate the effectiveness of this algorithm relative to other state-of-the-art MI techniques on a wide variety of benchmark datasets.

In this study, MILES was generalized to allow the use of alternative base learners in place of the default 1-norm support vector machine used by Chen et al (2006). Although MILES is not guaranteed to learn a weight-based model when non-linear base learners are used, the instance-based feature-space mapping performed by the algorithm ensures that these models are closely related to those that learn weights. The base learner must implicitly determine the importance of the transformed-space attributes, which correspond to points in instance space, thereby determining the importance of different parts of the instance space.

It was found that the 1-norm support vector machine recommended by [Chen et al., 2006] was not generally superior to the standard 2-norm SVM as a base learner for MILES. Though the 1-norm SVM was competitive as a MILES base learner, boosted decision stumps were the best performing base learner overall. Although MILES was competitive with all other MI algorithms tried in the experiment, the simpler MIWrapper and SimpleMI algorithms generally performed just as well as MILES when appropriate base learners were used.

Two new generalized MI assumptions were presented, which were designed to model the level of influence that instances have on bag-level class labels via a weight

function over instance space. The *weighted collective MI assumption* extends the *collective MI assumption* [Xu, 2003] by incorporating a weight function into the model, while the *weighted linear threshold MI assumption* is inspired by linear classification in single-instance learning. The precise relationship between the two assumptions was shown.

New algorithms were designed for learning MI concepts under the new assumptions. The IFLIW algorithm is an extension of the MIWrapper method which learns a weight function as well as the class probability function learnt by the original MIWrapper algorithm. YARDS is a variant of MILES which learns weighted linear threshold MI concepts when a linear base learner is used. Unlike MILES, IFLIW and YARDS (with a linear base classifier) each learn a weight function over instance space that is not bag-dependent.

The utility of IFLIW and YARDS was demonstrated on artificial data generated under the corresponding MI assumptions. Although MIWrapper was competitive in terms of classification accuracy on the artificial data, IFLIW and YARDS were able to achieve significantly lower root mean squared error rates on data generated according to their underlying MI assumptions. The new algorithms were also evaluated on real-world datasets. It was found that IFLIW and YARDS were not generally superior to MIWrapper and MILES on the real-world problems, although their performance was very competitive against those algorithms. Furthermore, IFLIW achieved 93.5% accuracy on the *suramin* dataset, which was a higher accuracy than those exhibited by the other algorithms tried in the experiments by a large margin, although this difference was not statistically significant according to the corrected resampled t-test.

7.1 Future Work

Many areas for future work were identified during the creation of this thesis. These can be loosely grouped into four categories: algorithm evaluation, learning instance weights, generalized MI and standard MI learning.

Algorithm Evaluation

Extensive experiments were performed to evaluate MILES, IFLIW and YARDS in Chapters 4 and 6. Due to time constraints and restrictions on the availability of computational resources, there were some limitations to these studies, however. In particular, parameter selection was not performed for the σ parameter for MILES and YARDS. It would be interesting to investigate the effect of this parameter on the performance of those algorithms.

[Chen et al., 2006] claim that attribute selection is important for MILES (and therefore, presumably, YARDS also) because of the high-dimensional feature space created by the algorithm. Although the competitive results for MILES with the 2-norm support vector machine relative to the default 1-norm SVM (which performs more aggressive attribute selection) indicate that attribute selection may not be crucial, this is still a potential area for future investigation.

In all experiments for IFLIW, the regression base learner was set to additive regression with decision stumps. Other regression learners may potentially be more effective, however. A study into the performance of the algorithm using different regression base learners would be beneficial.

Learning Instance Weights

The IFLIW algorithm introduced in Chapter 5 uses a heuristic approach to learn a weight function. This heuristic is not guaranteed to be optimal, so there is scope for the investigation of other approaches for learning weight functions within the IFLIW framework. Possibilities include the use of simulated annealing, genetic algorithms or other optimization techniques. Another approach would be to use the diverse density value [Maron and Lozano-Pérez, 1998] directly to determine the importance of different regions of instance space.

As MILES learns weights for the training instances, one way to learn a weight function for IFLIW is to build a regression algorithm on the weights predicted by MILES. We name this algorithm *MILES for Learning Instance Weights* (MIFLIW). YARDS can also be used to learn a weight function for IFLIW. However, as YARDS learns a weight function over the entire instance space, a regression model is not required in order to generalize the weight function to future instances; the YARDS

weight function can be used directly. This algorithm is named *YARDS for Learning Instance Weights* (YARFLIW). Some preliminary work has already been done for these approaches.

YARDS is a version of the MILES algorithm that uses a different feature space transformation to represent multi-instance bags in a propositional form. It was shown that the YARDS algorithm learns *weighted linear threshold* MI concepts when a linear base learner is used. The experimental results for YARDS demonstrate that alternative feature space transformations for MILES can be effective, and may be more appropriate for certain problem domains. Other alternative transformations may also prove to be useful.

The MILES algorithm uses the *most-likely cause* [Maron, 1998] diverse density estimator to compute the feature space transformation; obvious possible alternative include the other diverse density estimators defined by Maron including the *noisy-or* and *all-or-nothing* estimators. Another possibility is to find the centre of mass for a bag, and compute the mapping based on the distance between the centre of mass and the target point. The choice of feature space transformation determines the information available to the single-instance base learner, and thus largely determines the types of MI concepts that can be learnt. The exploration of alternative transformations for specific problem domains is a promising area for future research.

The YARDS method may also be effective for instance classification, which is important in some MI problem domains such as object detection [Chen et al., 2006]. Chen et al. provide a method for classifying instances within multi-instance bags via the bag-dependent weight function used by MILES. However, this method is only defined for a certain index set of instances in a bag — those which are the closest to one of the candidate target points and are hence able to contribute to the bag-level class label. This means that MILES is not able to classify all instances in a bag. The classification step is also bag dependent, which is counter-intuitive to the notion that a point in instance space has a specific label regardless of the bag that it belongs to. In contrast, YARDS learns a bag-independent instance-level classification function over the entire instance space. In fact, any algorithm that explicitly learns weighted linear threshold concepts must also learn such a classification function.

In Section 5.4.3, the *extended deterministic weighted collective MI assumption* was briefly introduced. This assumption extends the deterministic weighted col-

lective assumption to include a bias parameter that determines the location of the classification decision boundary. A future task is to develop algorithms that learn this type of MI concept. A simple approach would be to create a version of IFLIW that also predicts the bias parameter. Such an extension is also potentially applicable to the MIWrapper algorithm.

Generalized MI Assumptions and Algorithms

MIWrapper and IFLIW use a very simple heuristic for predicting instance-level class labels: each instance is merely assigned the class label of its parent bag. Although this simple approach works surprisingly well in practice, it may be possible to improve upon this with a more sophisticated instance labeling method.

An important area for future research is the further investigation of generalized MI assumptions that determine the relationship between instances and bag-level concepts. This thesis presented MI assumptions that used the notion of instance weights to determine the importance of each instance in bag-level classification. Another possibility is to define MI concepts in terms of instance-level concept concurrency relationships. For example, *beach scene* concepts can be defined as images which contain instances belonging to the *ocean* concept and the *sand* concept.

The ConMIL algorithm [Qi et al., 2007] learns such concurrency relationships at training time, but the concurrency information is only indirectly used at classification time, as classification decisions are made according to the standard MI assumption. This algorithm could potentially be improved by making use of concurrency relationships at classification time. Such a modification to the algorithm requires the abandonment of the standard MI assumption in favour of a generalized assumption that defines MI concepts in terms of concurrency relationships.

Multi-instance multi-label learning (MIML) [Zhou and Zhang, 2006] is a variant of multi-instance learning where each bag has a set of labels that apply to it, rather than just a single label. The standard MI assumption does not account for the possibility of multiple labels, and therefore is not directly applicable in this scenario. Zhou and Zhang upgrade the multiple-instance logistic regression algorithm [Xu and Frank, 2004] to learn MIML concepts, and thus implicitly use a multi-label version of the collective MI assumption. They do not precisely define the generalized MI assumptions that their algorithms rely upon, however. Much work remains to be

done to clarify the MI assumptions that are applicable in multi-instance multi-label learning.

Standard MI Learning

The maxDD algorithm [Maron and Lozano-Pérez, 1998] searches for target points by optimizing the diverse density function over instance space. To attempt to find the point in instance space with the maximum diverse density value, a quasi-Newtonian gradient ascent search is repeatedly applied, starting from each instance inside a positive training bag. The point with the maximum diverse density value is assumed to be the location of the target concept, and instance-level class probabilities are computed based on the distance to this target point. Bag-level classification decisions are made according to the standard MI assumption. However, the repeated optimization step is very computationally expensive. [Zhang and Goldman, 2002] presented a more efficient way to search for diverse density maxima using an expectation-maximization approach.

Another possible approach is to abandon the quasi-Newtonian optimization procedure used by maxDD, and instead only consider instances belonging to positive training bags when searching for the diverse density maxima. The potential effectiveness of such an approach is demonstrated by the diverse density-based algorithm MILES, which also uses the heuristic of considering only training instances when selecting candidate target points. MILES performs just as well as its predecessor DD-SVM, which (as in maxDD) performs gradient ascent searches to find candidate target points at diverse density local maxima.

7.2 Summary

Multi-instance learning is a variant of supervised machine learning where each learning example, called a bag, may contain multiple feature vectors. When a generalized view of multi-instance learning is taken, there are many possible ways that instances within a bag can interact to produce an overall class label for that example. This thesis explored the case where each instance is assigned a weight that determines its contribution to the label of its parent bag. An existing MI algorithm related to this scenario was identified, and was thoroughly evaluated via an empirical study. Precise

formalizations of the interactions between instances and bag labels were provided, and algorithms were presented that were designed to learn these types of multiple-instance concepts. The effectiveness of the new algorithms was demonstrated on both artificial data and real-world problems.

The field of MI learning is relatively young, and much work remains to be done in this area. A number of possible avenues for future research were identified in this thesis. In particular, the relationships between instances in a bag and the label of that bag differ between problem domains, and are often not well understood. The branch of multi-instance learning known as *generalized* MI is concerned with understanding the nature of these relationships, and developing algorithms that can learn effectively when different relationships are applicable. The development of generalized MI models for real-world problems, and algorithms to solve those problems, remains an important area for future research.

Appendix: Detailed Experimental Results for the New Algorithms

The full results for the experiments described in Chapter 6 were too numerous to be displayed in that chapter. To avoid disrupting the flow of the document, those results are instead provided in this appendix.

Experimental results for each single-instance base learner are shown for the MILES, IFLIW and YARDS algorithms, along with MIWrapper as a baseline method. All results were computed via 10×10-fold cross-validation. Significant differences versus MIWrapper were computed using a pairwise corrected t-test with significance level $\alpha = 0.05$. Tables A.1 — A.9 display percentage accuracy, while Tables A.10 — A.18 show the root mean squared error results.

Complete results for IFLIW using the 2-norm SVM with the RBF kernel on the *thioredoxin* dataset were not available at the time of submission. The partial results are displayed in the tables.

Table A.1: MILES, IFLIW and YARDS vs MIWrapper: C4.5 Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	84.3± 11.7	84.1±11.9	81.5±12.2	77.6±12.6
musk2	80.1± 11.3	82.5±12.1	80.1±12.0	67.2±14.2 •
eastwest	52.0± 35.5	50.0± 0.0	44.5±35.5	65.0±30.6
westeast	49.5± 32.2	50.0± 0.0	44.5±35.5	65.0±30.6
mutagenesis-atoms	76.4± 8.3	80.8± 8.1	80.9± 8.5	66.8± 2.7 •
mutagenesis-bonds	80.7± 8.9	77.1± 9.8	83.8± 8.1	80.5± 9.5
mutagenesis-chains	84.9± 7.2	79.3± 9.5	84.1± 8.4	80.2± 9.0
suramin	65.0± 45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	87.8± 2.7	84.3± 7.1	87.3± 3.8	86.0± 6.6
elephant	80.3± 7.7	77.5± 9.2	77.8± 8.8	65.4± 9.9 •
fox	64.1± 10.6	56.8±11.2	64.1± 9.8	49.7± 5.1 •
tiger	77.4± 9.2	69.7± 9.3	73.4± 9.4	60.0± 9.6 •
bikes	78.8± 4.6	72.5± 5.7 •	76.3± 5.2	72.7± 5.6 •
cars	67.7± 5.3	62.6± 4.7 •	68.2± 4.5	62.6± 5.3
people	78.4± 4.2	69.8± 5.8 •	76.3± 4.5	68.9± 5.5 •

◦, • statistically significant improvement or degradation vs MIWrapper

Table A.2: MILES, IFLIW and YARDS vs MIWrapper: Random Forests Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	87.3± 10.8	87.0±11.4	85.7±12.0	84.4±10.9
musk2	81.0± 11.4	81.7±11.2	79.7±12.1	69.4±13.8 •
eastwest	54.0± 31.5	80.0±24.6	71.0±31.1	72.5±28.8
westeast	55.0± 31.4	80.0±24.6	69.0±31.6	72.5±30.5
mutagenesis3-atoms	81.9± 8.5	82.0± 8.2	82.0± 8.1	81.9± 8.5
mutagenesis3-bonds	83.1± 8.7	79.7±10.5	82.6± 8.8	82.6± 8.4
mutagenesis3-chains	84.2± 7.5	80.4± 9.2	83.3± 8.1	79.6± 9.3
suramin	65.0± 45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	87.9± 2.6	87.7± 2.7	87.2± 2.5	87.4± 3.0
elephant	87.1± 6.9	82.3± 8.2	82.1± 8.7 •	66.6±10.3 •
fox	64.6± 9.6	64.9±10.2	64.1± 9.9	54.6±10.8 •
tiger	84.3± 8.1	78.6± 9.0 •	81.3± 8.8	62.0±10.0 •
bikes	82.2± 4.2	79.2± 4.4 •	80.2± 4.6	76.8± 4.8 •
cars	74.8± 4.6	71.7± 4.0	72.4± 4.8	67.0± 4.6 •
people	82.6± 4.0	77.5± 4.3 •	77.9± 4.3 •	73.0± 5.0 •

◦, • statistically significant improvement or degradation vs MIWrapper

Table A.3: MILES, IFLIW and YARDS vs MIWrapper: Adaboost + C4.5 Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	86.8± 11.1	85.8±12.0	86.8±11.1	80.3±11.9
musk2	82.7± 11.8	83.2±11.3	82.7±11.8	71.1±14.5
eastwest	56.0± 33.5	50.0± 0.0	47.5±35.8	64.5±31.2
westeast	57.5± 34.4	50.0± 0.0	47.5±35.8	64.5±31.2
mutagenesis-atoms	77.4± 8.1	79.5± 8.5	80.3± 8.6	67.1± 3.2 •
mutagenesis-bonds	82.2± 8.3	80.1± 9.9	81.4± 8.1	80.3± 8.6
mutagenesis-chains	85.1± 7.8	80.8± 8.1	83.0± 7.2	79.4± 8.3
suramin	65.0± 45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	88.0± 2.6	85.6± 6.4	88.5± 3.2	87.8± 5.9
elephant	84.2± 8.1	81.5± 8.9	84.4± 7.8	67.1±10.3 •
fox	62.6± 9.8	59.4±11.6	62.7± 9.7	49.7± 5.1 •
tiger	81.0± 8.1	75.4± 9.3	81.1± 8.0	61.5± 9.9 •
bikes	81.8± 4.5	78.0± 4.5 •	81.3± 4.2	75.2± 5.5 •
cars	71.4± 4.9	69.3± 5.0	71.4± 4.9	65.7± 5.1 •
people	81.5± 4.0	75.4± 4.8 •	81.1± 4.5	72.8± 4.6 •

◦, • statistically significant improvement or degradation vs MIWrapper

Table A.4: MILES, IFLIW and YARDS vs MIWrapper: Adaboost with Decision Stump Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	84.7± 10.7	88.0±11.6	82.6±12.5	82.7±12.1
musk2	79.7± 10.6	83.2±11.5	75.6±12.9	72.2±12.0
eastwest	69.0± 26.4	81.0±24.4	51.0±31.8	70.0±29.3
westeast	69.0± 26.4	81.0±24.4	51.0±31.8	70.0±29.3
mutagenesis-atoms	66.5± 2.3	83.9± 8.6 ○	73.3± 8.7 ○	78.6± 8.9 ○
mutagenesis-bonds	73.2± 8.4	86.3± 7.4 ○	79.9± 9.6	81.7± 8.4 ○
mutagenesis-chains	74.0± 7.7	86.0± 8.0 ○	73.3± 9.5	79.3± 9.1
suramin	65.0± 45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	87.1± 2.4	89.3± 4.0	86.7± 3.3	89.1± 4.9
elephant	85.5± 7.3	80.9± 7.7	82.9± 8.5	66.0±10.5 ●
fox	65.7± 9.6	61.6±10.9	63.3±11.0	52.1±11.0 ●
tiger	81.8± 8.5	80.5± 8.9	77.7± 9.1	66.4±10.9 ●
bikes	79.2± 4.6	78.0± 5.0	79.7± 5.6	76.2± 5.1
cars	71.3± 4.8	71.6± 4.1	71.4± 4.9	66.1± 4.0 ●
people	79.5± 4.3	75.6± 4.6 ●	76.9± 4.5	74.9± 4.5 ●

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.5: MILES, IFLIW and YARDS vs MIWrapper: Bagging with C4.5 Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	86.6± 11.0	86.0±11.5	84.1±12.6	80.7±12.5
musk2	81.5± 11.6	83.7±11.5	82.6±11.9	72.5±13.4
eastwest	54.0± 34.6	50.5± 5.0	54.0±38.7	69.0±27.3
westeast	53.5± 35.0	50.5± 5.0	54.0±38.7	69.0±27.3
mutagenesis-atoms	79.7± 8.2	80.5± 7.7	79.1± 9.2	75.5± 9.0
mutagenesis-bonds	82.9± 8.5	77.4± 8.9	83.9± 8.1	80.1± 9.0
mutagenesis-chains	85.3± 7.6	79.8± 9.1	82.9± 8.1	80.3± 8.5
suramin	65.0± 45.2	62.0±46.1	65.0±45.2	62.0±46.1
thioredoxin	87.9± 2.6	88.2± 4.6	87.0± 2.4	89.7± 5.1
elephant	84.1± 7.7	84.0± 8.3	77.7±10.4 ●	68.1±10.1 ●
fox	65.7± 8.8	61.4±10.3	62.8±10.1	52.9±10.1 ●
tiger	81.7± 9.4	75.7± 8.4	75.9± 9.2 ●	65.5±11.0 ●
bikes	82.5± 4.0	77.7± 5.1 ●	80.6± 4.4	76.2± 5.2 ●
cars	74.3± 4.5	70.5± 4.9 ●	70.3± 4.7 ●	66.2± 4.2 ●
people	81.8± 3.9	76.6± 4.7 ●	77.4± 4.9 ●	73.2± 5.4 ●

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.6: MILES, IFLIW and YARDS vs MIWrapper: 2-Norm SVM with Linear Kernel Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	86.9± 11.8	86.6± 9.9	85.1±11.6	79.7±12.2
musk2	81.2± 11.8	88.6±10.1	82.8±11.8	75.8±11.6
eastwest	53.0± 33.2	54.0±33.1	62.0±31.9	65.0±35.2
westeast	53.0± 33.2	54.5±32.6	62.0±31.9	65.0±35.2
mutagenesis-atoms	66.5± 2.3	81.5± 8.2 ○	66.3± 2.5	66.5± 2.4
mutagenesis-bonds	66.5± 2.3	81.3± 9.7 ○	71.6± 5.7 ○	79.0± 8.6 ○
mutagenesis-chains	68.0± 3.8	77.6± 8.2 ○	68.9± 5.0	80.5± 8.3 ○
suramin	35.0± 45.2	65.0±45.2	34.0±44.9	65.0±45.2
thioredoxin	87.1± 2.4	69.0±10.6 ●	87.1± 2.4	68.4±11.4 ●
elephant	84.5± 8.7	83.9± 8.0	80.4± 8.7	66.4±13.4 ●
fox	59.2± 9.5	61.8± 9.4	59.2±10.9	49.9±11.8
tiger	80.5± 8.0	80.8± 8.8	78.8± 9.6	57.4±10.4 ●
bikes	81.5± 4.1	80.5± 4.7	81.6± 4.9	79.4± 4.7
cars	70.8± 4.6	72.5± 4.9	70.2± 4.9	64.8± 6.3 ●
people	79.6± 4.3	74.9± 4.9 ●	76.3± 4.8 ●	76.9± 4.7

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.7: MILES, IFLIW and YARDS vs MIWrapper: 2-Norm SVM with RBF Kernel Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	80.8± 12.2	76.1±14.6	81.2±12.6	73.7±13.4
musk2	82.8± 11.8	76.3±11.7	80.0±12.7	63.3±11.3 ●
eastwest	68.0± 29.7	80.0±24.6	55.0±32.2	68.5±24.3
westeast	68.0± 29.7	80.0±24.6	54.5±31.9	68.5±24.3
mutagenesis-atoms	66.5± 2.3	82.7± 8.8 ○	66.5± 3.0	68.5± 5.7
mutagenesis-bonds	66.5± 2.3	79.1± 8.8 ○	66.7± 3.6	78.3± 8.7 ○
mutagenesis-chains	69.6± 3.5	76.6± 9.6 ○	71.4± 6.5	82.5± 8.6 ○
suramin	35.0± 45.2	65.0±45.2	93.5±19.7 ○	65.0±45.2
thioredoxin	87.1± 2.4	87.1± 2.4	87.0*± 2.7	84.0± 6.2
elephant	82.0± 8.4	52.8± 5.7 ●	80.3± 8.7	59.5±10.6 ●
fox	57.2± 10.4	54.8± 5.6	54.6±10.3	46.7±10.2 ●
tiger	81.0± 8.0	63.8± 6.1 ●	77.8± 9.2	57.0± 9.0 ●
bikes	76.8± 4.3	77.2± 4.5	77.8± 4.9	75.4± 4.7
cars	66.6± 5.1	71.0± 4.6 ○	66.6± 5.1	66.9± 4.9
people	76.0± 4.4	75.3± 4.5	74.1± 4.6 ●	74.9± 4.3

* execution did not complete in time for submission for IFLIW on the *thioredoxin* dataset. The displayed result is based on the 35/100 completed folds.
○, ● statistically significant improvement or degradation vs MIWrapper

Table A.8: MILES, IFLIW and YARDS vs MIWrapper: 1-Norm SVM Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	83.6± 11.7	83.2±11.7	80.8±12.7	80.6±11.4
musk2	78.6± 13.1	90.4± 9.3 ○	78.6±13.4	82.6±11.2
eastwest	47.0± 28.3	50.0± 0.0	56.5±23.2	65.0±35.2
westeast	47.5± 29.6	50.0± 0.0	56.5±23.2	65.0±35.2
mutagenesis-atoms	66.5± 2.3	77.4± 9.3 ○	66.5± 2.3	66.5± 2.3
mutagenesis-bonds	66.5± 2.3	73.1±13.0	66.5± 2.3	81.3± 8.9 ○
mutagenesis-chains	66.7± 2.6	77.0± 9.0 ○	66.6± 3.1	82.7± 9.7 ○
suramin	65.0± 45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	87.1± 2.4	88.1± 5.1	87.1± 2.4	86.3± 4.4
elephant	83.4± 8.0	84.3± 8.7	79.9± 9.7	82.9± 8.5
fox	58.9± 9.6	61.0± 9.5	59.4± 9.3	55.5± 9.5
tiger	77.7± 8.1	82.0± 8.5	75.0± 8.4	81.3± 8.8
bikes	81.9± 3.9	76.1± 5.0 ●	80.6± 4.0	83.9± 4.6
cars	72.3± 4.9	66.5± 5.5 ●	70.4± 4.8	71.4± 4.7
people	79.5± 4.3	71.1± 4.9 ●	75.8± 4.7 ●	78.5± 4.2

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.9: MILES, IFLIW and YARDS vs MIWrapper: Logistic Regression Base Learner (Classification Accuracy)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	78.9± 12.5	84.8±11.3	78.3±12.1	81.2±11.4
musk2	81.7± 12.7	85.8±11.0	82.4±13.3	79.7±12.8
eastwest	61.5± 33.2	64.5±29.6	48.0±31.7	65.0±35.2
westeast	61.5± 33.2	68.5±33.1	48.0±31.7	65.0±35.2
mutagenesis-atoms	66.5± 2.3	83.8± 7.2 ○	67.2± 5.2	66.8± 6.1
mutagenesis-bonds	67.2± 3.3	80.2± 8.8 ○	75.0± 8.2 ○	80.8± 8.5 ○
mutagenesis-chains	70.6± 6.5	73.5± 9.4	71.8± 7.7	83.9± 7.8 ○
suramin	65.0± 45.2	65.0±45.2	65.0±45.2	65.0±45.2
thioredoxin	87.1± 2.4	87.1*± 3.9	87.0± 2.7	86.1*± 4.0
elephant	84.0± 8.6	79.6± 9.1	79.1± 9.0	83.9± 8.7
fox	58.4± 10.2	63.6± 8.9	56.6± 9.8	55.9±10.5
tiger	78.4± 8.3	80.0± 9.2	74.6± 8.8	82.1± 8.5
bikes	82.2± 3.8	72.4± 4.8 ●	79.7± 5.1	82.8± 4.4
cars	71.8± 4.9	63.9± 4.9 ●	70.3± 5.0	72.18± 5.0
people	78.7± 4.2	66.9± 5.0 ●	73.1± 4.4 ●	78.28± 4.5

∗ We were unable to allocate enough memory to run MILES/YARDS + Logistic on the *thioredoxin* dataset. This result was obtained using an alternative logistic regression algorithm, implemented as *SimpleLogistic* in WEKA. The algorithm uses LogitBoost to learn a logistic regression model. See [Landwehr et al., 2003] for more information.

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.10: MILES, IFLIW and YARDS vs MIWrapper: C4.5 Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.33± 0.12	0.34±0.19	0.34±0.13	0.44±0.15
musk2	0.39± 0.11	0.37±0.18	0.38±0.12	0.53±0.14 ●
eastwest	0.52± 0.16	0.50±0.00	0.62±0.25	0.44±0.35
westeast	0.52± 0.16	0.50±0.00	0.62±0.25	0.44±0.35
mutagenesis-atoms	0.39± 0.04	0.38±0.07	0.38±0.08	0.47±0.01 ●
mutagenesis-bonds	0.37± 0.07	0.42±0.08	0.35±0.09	0.38±0.07
mutagenesis-chains	0.33± 0.07	0.39±0.09	0.34±0.09	0.39±0.08
suramin	0.46± 0.25	0.50±0.14	0.37±0.45	0.50±0.14
thioredoxin	0.31± 0.03	0.38±0.11	0.32±0.05	0.36±0.10
elephant	0.37± 0.06	0.44±0.10 ●	0.39±0.07	0.54±0.08 ●
fox	0.49± 0.06	0.61±0.08 ●	0.51±0.06	0.52±0.04
tiger	0.39± 0.07	0.53±0.08 ●	0.43±0.07	0.51±0.07 ●
bikes	0.39± 0.03	0.51±0.05 ●	0.41±0.04	0.51±0.05 ●
cars	0.46± 0.02	0.60±0.04 ●	0.49±0.03 ●	0.59±0.05 ●
people	0.39± 0.03	0.54±0.05 ●	0.40±0.03	0.54±0.05 ●

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.11: MILES, IFLIW and YARDS vs MIWrapper: Random Forests Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.30± 0.08	0.30±0.08	0.31±0.10	0.33±0.08
musk2	0.36± 0.08	0.35±0.07	0.36±0.09	0.45±0.09 ●
eastwest	0.51± 0.11	0.32±0.25 ○	0.39±0.24	0.40±0.25
westeast	0.51± 0.11	0.32±0.25 ○	0.39±0.25	0.40±0.25
mutagenesis-atoms	0.36± 0.07	0.38±0.07	0.36±0.07	0.39±0.09
mutagenesis-bonds	0.36± 0.07	0.38±0.07	0.36±0.07	0.36±0.08
mutagenesis-chains	0.34± 0.07	0.37±0.09	0.34±0.08	0.38±0.08
suramin	0.47± 0.21	0.51±0.14	0.44±0.32	0.51±0.14
thioredoxin	0.30± 0.03	0.30±0.04	0.32±0.03 ●	0.30±0.03
elephant	0.35± 0.04	0.36±0.05	0.36±0.05	0.47±0.06 ●
fox	0.47± 0.02	0.47±0.04	0.47±0.03	0.54±0.05 ●
tiger	0.37± 0.04	0.40±0.05 ●	0.36±0.06	0.49±0.06 ●
bikes	0.37± 0.02	0.38±0.03 ●	0.38±0.04	0.40±0.03 ●
cars	0.44± 0.01	0.43±0.02	0.44±0.03	0.47±0.02 ●
people	0.37± 0.02	0.39±0.02 ●	0.40±0.04 ●	0.41±0.02 ●

○, ● statistically significant improvement or degradation vs MIWrapper

Table A.12: MILES, IFLIW and YARDS vs MIWrapper: Adaboost with C4.5 Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.30± 0.13	0.31±0.21	0.30±0.13 •	0.41±0.16
musk2	0.35± 0.13	0.36±0.17	0.35±0.13 ○	0.51±0.14 •
eastwest	0.52± 0.17	0.50±0.00	0.59±0.28	0.46±0.37
westeast	0.52± 0.17	0.50±0.00	0.58±0.27	0.46±0.37
mutagenesis-atoms	0.39± 0.04	0.39±0.07	0.38±0.07	0.47±0.02 •
mutagenesis-bonds	0.37± 0.07	0.39±0.08	0.36±0.08	0.38±0.09
mutagenesis-chains	0.34± 0.08	0.38±0.08	0.35±0.08	0.40±0.10
suramin	0.47± 0.21	0.50±0.14	0.34±0.42	0.50±0.14
thioredoxin	0.31± 0.04	0.36±0.11	0.31±0.04	0.33±0.10
elephant	0.33± 0.07	0.40±0.11 •	0.33±0.07	0.53±0.08 •
fox	0.50± 0.06	0.60±0.09 •	0.50±0.05	0.51±0.02
tiger	0.36± 0.07	0.46±0.10 •	0.36±0.07	0.50±0.06 •
bikes	0.36± 0.03	0.45±0.05 •	0.36±0.03	0.48±0.05 •
cars	0.43± 0.02	0.53±0.04 •	0.43±0.02	0.56±0.04 •
people	0.37± 0.03	0.47±0.05 •	0.37±0.03	0.50±0.04 •

○, • statistically significant improvement or degradation vs MIWrapper

Table A.13: MILES, IFLIW and YARDS vs MIWrapper: Adaboost with Decision Stump Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.31± 0.09	0.26±0.21	0.33±0.14	0.36±0.16
musk2	0.37± 0.08	0.37±0.16	0.41±0.10	0.47±0.13 •
eastwest	0.47± 0.09	0.27±0.34	0.54±0.20	0.40±0.36
westeast	0.47± 0.09	0.27±0.34	0.54±0.20	0.40±0.36
mutagenesis-atoms	0.44± 0.02	0.35±0.10 ○	0.41±0.05	0.41±0.05
mutagenesis-bonds	0.41± 0.03	0.33±0.09 ○	0.37±0.06	0.37±0.08
mutagenesis-chains	0.40± 0.03	0.32±0.11 ○	0.40±0.05	0.38±0.08
suramin	0.55± 0.20	0.50±0.14	0.37±0.46	0.50±0.14
thioredoxin	0.33± 0.03	0.31±0.07	0.31±0.04	0.31±0.08
elephant	0.33± 0.06	0.40±0.08 •	0.36±0.08	0.47±0.05 •
fox	0.47± 0.03	0.54±0.07 •	0.49±0.06	0.52±0.04 •
tiger	0.36± 0.06	0.39±0.09	0.40±0.07	0.47±0.06 •
bikes	0.39± 0.02	0.40±0.04	0.38±0.03	0.40±0.04
cars	0.46± 0.01	0.44±0.03	0.44±0.02 ○	0.46±0.02
people	0.38± 0.02	0.42±0.04 •	0.40±0.03 •	0.41±0.03 •

○, • statistically significant improvement or degradation vs MIWrapper

Table A.14: MILES, IFLIW and YARDS vs MIWrapper: Bagging With C4.5 Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.29± 0.09	0.30±0.10	0.31±0.13	0.36±0.10
musk2	0.36± 0.10	0.34±0.11	0.35±0.12	0.43±0.10
eastwest	0.52± 0.12	0.50±0.00	0.53±0.32	0.41±0.26
westeast	0.52± 0.12	0.50±0.00	0.53±0.32	0.41±0.26
mutagenesis-atoms	0.37± 0.05	0.38±0.06	0.39±0.07	0.42±0.05 •
mutagenesis-bonds	0.36± 0.07	0.39±0.07	0.35±0.08	0.37±0.07
mutagenesis-chains	0.33± 0.07	0.36±0.07	0.35±0.08	0.37±0.07
suramin	0.46± 0.25	0.51±0.14	0.37±0.45	0.51±0.14
thioredoxin	0.30± 0.03	0.30±0.05	0.33±0.03 •	0.29±0.06
elephant	0.34± 0.05	0.35±0.07	0.40±0.09 •	0.46±0.06 •
fox	0.46± 0.04	0.49±0.05	0.53±0.07 •	0.51±0.03 •
tiger	0.37± 0.06	0.41±0.06 •	0.42±0.08 •	0.47±0.06 •
bikes	0.37± 0.02	0.39±0.04 •	0.37±0.04	0.40±0.03 •
cars	0.44± 0.01	0.44±0.02	0.46±0.03 •	0.47±0.03 •
people	0.37± 0.02	0.40±0.03 •	0.41±0.04 •	0.42±0.03 •

○, • statistically significant improvement or degradation vs MIWrapper

Table A.15: MILES, IFLIW and YARDS vs MIWrapper: 2-Norm SVM with Linear Kernel Base Learner(Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.30± 0.10	0.31±0.18	0.32±0.12	0.37±0.12 •
musk2	0.34± 0.11	0.27±0.17	0.34±0.11	0.43±0.10
eastwest	0.50± 0.12	0.57±0.35	0.48±0.18	0.43±0.17
westeast	0.50± 0.12	0.57±0.35	0.48±0.18	0.43±0.17
mutagenesis-atoms	0.47± 0.01	0.38±0.07 ○	0.47±0.01	0.47±0.01
mutagenesis-bonds	0.45± 0.01	0.38±0.08 ○	0.43±0.02 ○	0.38±0.06 ○
mutagenesis-chains	0.45± 0.01	0.40±0.08	0.45±0.02	0.35±0.06 ○
suramin	0.56± 0.11	0.50±0.14	0.68±0.41	0.50±0.14
thioredoxin	0.33± 0.03	0.54±0.10 •	0.33±0.03	0.54±0.11 •
elephant	0.34± 0.06	0.34±0.08	0.37±0.08	0.44±0.07 •
fox	0.50± 0.04	0.49±0.05	0.52±0.06 •	0.50±0.02
tiger	0.37± 0.06	0.36±0.08	0.40±0.07	0.48±0.03 •
bikes	0.37± 0.02	0.37±0.05	0.37±0.03	0.37±0.04
cars	0.46± 0.01	0.44±0.03	0.45±0.01	0.47±0.02
people	0.39± 0.02	0.42±0.04 •	0.42±0.02 •	0.40±0.03 •

○, • statistically significant improvement or degradation vs MIWrapper

Table A.16: MILES, IFLIW and YARDS vs MIWrapper: 2-Norm SVM with RBF Kernel Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.35± 0.08	0.45±0.19	0.36±0.09	0.39±0.09
musk2	0.33± 0.10	0.47±0.14 •	0.35±0.10	0.48±0.06 •
eastwest	0.46± 0.06	0.28±0.35	0.49±0.11	0.39±0.19
westeast	0.46± 0.06	0.28±0.35	0.49±0.11	0.39±0.19
mutagenesis-atoms	0.47± 0.01	0.40±0.12	0.47±0.01	0.47±0.03
mutagenesis-bonds	0.46± 0.01	0.45±0.10	0.47±0.02 •	0.39±0.07 ○
mutagenesis-chains	0.45± 0.01	0.47±0.10	0.45±0.03	0.36±0.08 ○
suramin	0.54± 0.09	0.37±0.46	0.36±0.10 ○	0.50±0.14
thioredoxin	0.32± 0.03	0.36±0.03 •	0.33*±0.00	0.36±0.06
elephant	0.37± 0.05	0.69±0.04 •	0.38±0.04	0.49±0.03 •
fox	0.49± 0.03	0.67±0.04 •	0.49±0.02	0.51±0.02
tiger	0.38± 0.05	0.60±0.05 •	0.40±0.06	0.49±0.03 •
bikes	0.39± 0.02	0.48±0.05 •	0.39±0.03	0.40±0.03
cars	0.47± 0.01	0.54±0.04 •	0.47±0.01	0.46±0.02
people	0.41± 0.02	0.50±0.04 •	0.43±0.02 •	0.41±0.03

* execution did not complete in time for submission for IFLIW on the *thioredoxin* dataset. The displayed result is based on the 35/100 completed folds.
○, • statistically significant improvement or degradation vs MIWrapper

Table A.17: MILES, IFLIW and YARDS vs MIWrapper: 1-Norm SVM Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.35± 0.11	0.34±0.06	0.37±0.13	0.38±0.08
musk2	0.38± 0.11	0.29±0.07 ○	0.38±0.12	0.38±0.10
eastwest	0.51± 0.09	0.55±0.00	0.51±0.15	0.42±0.14
westeast	0.51± 0.09	0.55±0.00	0.51±0.15	0.42±0.14
mutagenesis-atoms	0.47± 0.01	0.41±0.05 ○	0.47±0.01 •	0.48±0.01 •
mutagenesis-bonds	0.46± 0.02	0.41±0.05 ○	0.47±0.02	0.37±0.06 ○
mutagenesis-chains	0.45± 0.02	0.40±0.05 ○	0.46±0.02	0.36±0.08 ○
suramin	0.44± 0.21	0.44±0.21 ○	0.44±0.21 ○	0.44±0.21 ○
thioredoxin	0.36± 0.02	0.34±0.03	0.36±0.02 •	0.36±0.03
elephant	0.36± 0.06	0.36±0.06	0.38±0.09	0.35±0.07
fox	0.51± 0.05	0.49±0.04	0.55±0.06 •	0.55±0.06 •
tiger	0.39± 0.06	0.37±0.06	0.43±0.07 •	0.38±0.08
bikes	0.38± 0.02	0.40±0.03 •	0.38±0.03	0.35±0.04 ○
cars	0.45± 0.01	0.48±0.03 •	0.45±0.02	0.44±0.03
people	0.39± 0.02	0.44±0.03 •	0.42±0.03 •	0.39±0.03

○, • statistically significant improvement or degradation vs MIWrapper

Table A.18: MILES, IFLIW and YARDS vs MIWrapper: Logistic Regression Base Learner (Root Mean Squared Error)

Dataset	MIWrapper	MILES	IFLIW	YARDS
musk1	0.40± 0.12	0.33±0.19	0.40±0.11	0.39±0.16
musk2	0.35± 0.12	0.32±0.19	0.36±0.14	0.41±0.19
eastwest	0.47± 0.12	0.46±0.35	0.59±0.25	0.43±0.17
westeast	0.47± 0.12	0.41±0.39	0.59±0.25	0.43±0.17
mutagenesis-atoms	0.46± 0.01	0.38±0.09 ○	0.45±0.03	0.44±0.02 ○
mutagenesis-bonds	0.44± 0.02	0.42±0.09	0.42±0.04	0.37±0.06 ○
mutagenesis-chains	0.42± 0.02	0.49±0.09 ●	0.43±0.04	0.34±0.06 ○
suramin	0.50± 0.23	0.50±0.14	0.35±0.44	0.50±0.14
thioredoxin	0.33± 0.03	0.37*±0.09	0.32±0.03	0.45*±0.08 ●
elephant	0.35± 0.06	0.42±0.10 ●	0.39±0.08 ●	0.33±0.09
fox	0.52± 0.05	0.59±0.07 ●	0.57±0.06 ●	0.56±0.06 ●
tiger	0.38± 0.06	0.42±0.11	0.44±0.08 ●	0.38±0.08
bikes	0.36± 0.02	0.51±0.05 ●	0.38±0.03 ●	0.36±0.04
cars	0.45± 0.01	0.58±0.04 ●	0.46±0.02	0.44±0.00
people	0.39± 0.02	0.56±0.04 ●	0.44±0.03 ●	0.41±0.00

∗ We were unable to allocate enough memory to run MILES/YARDS + Logistic on the *thioredoxin* dataset. This result was obtained using an alternative logistic regression algorithm, implemented as *SimpleLogistic* in WEKA. The algorithm uses LogitBoost to learn a logistic regression model. See [Landwehr et al., 2003] for more information.

○, ● statistically significant improvement or degradation vs MIWrapper

Bibliography

[Andrews et al., 2002] Andrews, S., Tsochantaridis, I., and Hofmann, T. (2002). Support vector machines for multiple-instance learning. *Advances in Neural Information Processing Systems*, 15.

[Asuncion and Newman, 2007] Asuncion, A. and Newman, D. (2007). UCI machine learning repository.

[Auer and Ortner, 2004] Auer, P. and Ortner, R. (2004). A boosting approach to multiple instance learning. In *ECML*, pages 63–74.

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

[Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Buciluǎ et al., 2006] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, New York, NY, USA. ACM.

[Burl et al., 1998] Burl, M. C., Weber, M., and Perona, P. (1998). A probabilistic approach to object recognition using local photometry and global geometry. In *Computer Vision – ECCV 98*, volume 1407/1998 of *Lecture Notes in Computer Science*, pages 628–641.

[Carson et al., 1999] Carson, C., Thomas, M., Belongie, S., Hellerstein, J. M., and Malik, J. (1999). Blobworld: A system for region-based image indexing and

retrieval. In *Third International Conference on Visual Information Systems*. Springer.

[Chen et al., 2006] Chen, Y., Bi, J., and Wang, J. Z. (2006). Miles: Multiple-instance learning via embedded instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):1931–1947.

[Chen and Wang, 2004] Chen, Y. and Wang, J. Z. (2004). Image categorization by learning and reasoning with regions. *Journal of Machine Learning Research*, 5:913–939.

[Chevaleyre and Zucker, 2001] Chevaleyre, Y. and Zucker, J.-D. (2001). Solving multiple-instance and multiple-part learning problems with decision trees and rule sets. application to the mutagenesis problem. In Stroulia, E. and Matwin, S., editors, *Canadian Conference on AI*, volume 2056 of *Lecture Notes in Computer Science*, pages 204–214. Springer.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[De Raedt, 1998] De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: The missing links. In *Proceedings of the Eight International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*.

[Dietterich and Bakiri, 1995] Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286.

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

[Dong, 2006] Dong, L. (2006). A comparison of multi-instance learning algorithms. Master's thesis, University of Waikato.

[Edgar, 1990] Edgar, G. A. (1990). *Measure, Topology, and Fractal Geometry (2nd Edition)*. Undergraduate Texts in Mathematics. Springer-Verlag, New York.

[Frank and Xu, 2003] Frank, E. and Xu, X. (2003). Applying propositional learning algorithms to multi-instance data. Technical report, Department of Computer Science, University of Waikato.

[Freund and Schapire, 1996] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156.

[Friedman et al., 1998] Friedman, J., Hastie, T., and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University.

[Gärtner et al., 2002] Gärtner, T., Flach, P. A., Kowalczyk, A., and Smola, A. (2002). Multi-instance kernels. In *Proc. 19th Int. Conf. on Machine Learning*, pages 179–186, San Francisco, CA. Morgan Kaufmann.

[Haussler, 1999] Haussler, D. (1999). Convolution kernels on discrete structures. Technical report, UC Santa Cruz.

[Hsu et al., 2002] Hsu, W., Lee, M. L., and Zhang, J. (2002). Image mining: Trends and developments. *J. Intell. Inf. Syst.*, 19(1):7–23.

[King et al., 1993] King, R., Srinivasan, A., Muggleton, S., Feng, C., Lewis, R., and Sternberg, M. (1993). Drug design using inductive logic programming. In *Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences*, volume 1, pages 646–655.

[Kohavi and John, 1997] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

[Krogel and Wrobel, 2002] Krogel, M.-A. and Wrobel, S. (2002). Feature selection for propositionalization. In *Discovery Science: 5th International Conference*, pages 430–434. Springer.

[Landwehr et al., 2003] Landwehr, N., Hall, M., and Frank, E. (2003). Logistic model trees. In *Proc 14th European Conference on Machine Learning*, Cavtat-Dubrovnik, Croatia, pages 241–252. Springer.

[Larson and Michalski, 1977] Larson, J. and Michalski, R. S. (1977). Inductive inference of vl decision rules. workshop in pattern-directed inference systems. In *SIGART Newsletter*, volume 63, pages 38–44. ACM.

[le Cessie and van Houwelingen, 1992] le Cessie, S. and van Houwelingen, J. (1992). Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201.

[Littlestone, 1987] Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318.

[Manjunath and Ma, 1996] Manjunath, B. S. and Ma, W.-Y. (1996). Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842.

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

[Maron and Lozano-Pérez, 1998] Maron, O. and Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

[Maron and Ratan, 1998] Maron, O. and Ratan, A. L. (1998). Multiple-instance learning for natural scene classification. In *Proc. 15th International Conf. on Machine Learning*, pages 341–349. Morgan Kaufmann, San Francisco, CA.

[Mayo, 2007] Mayo, M. (2007). Effective classifiers for detecting objects. In *Proceedings of the Fourth International Conference on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS 07).*

[Michie et al., 1994] Michie, D., Muggleton, S., Page, D., and Srinivasan, A. (1994). To the international computing community: A new East-West challenge. Technical report, Oxford University Computing laboratory, Oxford,UK.

[Microsoft Game Studios, 1990] Microsoft Game Studios (1990). Chip's challenge. Microsoft Entertainment Pack.

[Murray et al., 2005] Murray, J. F., Hughes, G. F., and Kreutz-Delgado, K. (2005). Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 6:783–816.

[Nadeau and Bengio, 2003] Nadeau, C. and Bengio, Y. (2003). Inference for the Generalization Error. *Machine Learning*, 52(3):239–281.

[Nigam and Ghani, 2000] Nigam, K. and Ghani, R. (2000). Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 86–93. ACM Press New York, NY, USA.

[Opelt et al., 2006] Opelt, A., Pinz, A., Fussenegger, M., and Auer, P. (2006). Generic object recognition with boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(3):416–431.

[Platt, 1998] Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, Redmon, Washington.

[Qi et al., 2007] Qi, G., Hua, X., Rui, Y., Mei, T., Tang, J., and Zhang, H. (2007). Concurrent multiple instance learning for image categorization. In *Proceeding of IEEE Conference on Computer Vision and Pattern Recognition 2007*, pages 1–8.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

[Quinlan, 1986] Quinlan, R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Ray and Craven, 2005] Ray, S. and Craven, M. (2005). Supervised learning versus multiple instance learning: an empirical comparison. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 697–704. Omnipress.

[Reutemann, 2004] Reutemann, P. (2004). Development of a propositionalization toolbox. Master's thesis, Albert Ludwigs University of Freiburg.

[Reutemann et al., 2004] Reutemann, P., Pfahringer, B., and 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*, volume 3339 of *Lecture Notes in Computer Science*, pages 1017–1023. Springer Berlin.

[Schapire and Singer, 2000] Schapire, R. E. and Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

[Scott et al., 2005] Scott, S., Zhang, J., and Brown, J. (2005). On generalized multiple-instance learning. *International Journal of Computational Intelligence and Applications*, 5(1):21–35.

[Srinivasan et al., 1994] Srinivasan, A., Muggleton, S., King, R., and Sternberg, M. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In Wrobel, S., editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH.

[Tao and Scott, 2004] Tao, Q. and Scott, S. (2004). A faster algorithm for generalized multiple-instance learning. In *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*, pages 550–555.

[Tao et al., 2004a] Tao, Q., Scott, S., Vinodchandran, N., and Osugi, T. T. (2004a). Svm-based generalized multiple-instance learning via approximate box counting. In *Proceedings of the 21st International Conference on Machine Learning*, pages 779–806.

[Tao et al., 2004b] Tao, Q., Scott, S., Vinodchandran, N. V., Osugi, T., and Mueller, B. (2004b). An extended kernel for generalized multiple-instance learning. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 272–277.

[Wang et al., 2004] Wang, C., Scott, S., Zhang, J., Tao, Q., Fomenko, D., and Gladyshev, V. (2004). A study in modeling low-conservation protein superfamilies. Technical report, Department of Comp. Sci., University of Nebraska-Lincoln.

[Wang and Zucker, 2000] Wang, J. and Zucker, J.-D. (2000). Solving the multiple-instance problem: A lazy learning approach. In *Proc. 17th International Conf. on Machine Learning*, pages 1119–1125. Morgan Kaufmann, San Francisco, CA.

[Weidmann, 2003] Weidmann, N. (2003). Two-level classification for generalized multi-instance data. Master's thesis, Albert Ludwigs University of Freiburg.

[Weidmann et al., 2003] Weidmann, N., Frank, E., and Pfahringer, B. (2003). A two-level learning method for generalized multi-instance problems. In *Machine Learning: ECML 2003*, Lecture Notes in Computer Science, pages 468–479. Springer Berlin.

[Witten and Frank, 2005] Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques (2nd Edition)*. Morgan Kaufmann, San Francisco.

[Xu, 2003] Xu, X. (2003). Statistical learning in multiple instance problems. Master's thesis, University of Waikato.

[Xu and Frank, 2004] Xu, X. and Frank, E. (2004). Logistic regression and boosting for labeled bags of instances. In Dai, H., Srikant, R., and Zhang, C., editors, *Proc Eighth Pacific-Asia Advances in Knowledge Discovery and Data Mining Conference (PAKDD)*, pages 272–281. Springer-Verlag, Berlin.

[Yager, 1980] Yager, R. (1980). On a general class of fuzzy connectives. *Fuzzy Sets Systems*, 4(3):235–242.

[Zhang and Goldman, 2002] Zhang, Q. and Goldman, S. (2002). EM-DD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems 14: Proceedings of the 2002 Conference*. MIT Press.

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

[Zhou and Zhang, 2006] Zhou, Z.-H. and Zhang, M.-L. (2006). Multi-instance multi-label learning with application to scene classification. In Schölkopf, B.,

Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems*, pages 1609–1616. MIT Press.

[Zhu et al., 2003] Zhu, J., Rosset, S., Hastie, T., and Tibshirani, R. (2003). 1-norm support vector machines. Technical report, Stanford University.