Statistical Learning in Multiple Instance Problems

Xin Xu

A thesis submitted in partial fulfilment of

the requirements for the degree of

Master of Science

at the

University of Waikato

Department of Computer Science

Hamilton, New Zealand

June 2003

Abstract

Multiple instance (MI) learning is a relatively new topic in machine learning. It is concerned with supervised learning but differs from normal supervised learning in two points: (1) it has multiple instances in an example (and there is only one instance in an example in standard supervised learning), and (2) only one class label is observable for all the instances in an example (whereas each instance has its own class label in normal supervised learning). In MI learning there is a common assumption regarding the relationship between the class label of an example and the "unobservable" class labels of the instances inside it. This assumption, which is called the "MI assumption" in this thesis, states that "An example is positive if at least one of its instances is positive and negative otherwise".

In this thesis, we first categorize current MI methods into a new framework. According to our analysis, there are two main categories of MI methods, instance-based and metadata-based approaches. Then we propose a new assumption for MI learning, called the "collective assumption". Although this assumption has been used in some previous MI methods, it has never been explicitly stated,[1] and this is the first time that it is formally specified. Using this new assumption we develop new algorithms — more specifically two instance-based and one metadata-based methods. All of these methods build probabilistic models and thus implement statistical learning algorithms. The exact generative models underlying these methods are explicitly stated and illustrated so that one may clearly understand the situations

[1] As a matter of fact, for some of these methods, it is actually claimed that they use the standard MI assumption stated above.

to which they can best be applied. The empirical results presented in this thesis show that they are competitive on standard benchmark datasets. Finally, we explore some practical applications of MI learning, both existing and new ones.

This thesis makes three contributions: a new framework for MI learning, new MI methods based on this framework and experimental results for new applications of MI learning.

Acknowledgements

There are a number of people I want to thank for their help with my thesis.

First of all, my supervisor, Dr. Eibe Frank. I do not really know what I can say to express my gratitude. He contributed virtually all the ideas involved in this thesis. What is more important, he always provided me with his support for my work. When I was puzzled by a derivation or analysis of an algorithm, he was always the one to help me sort it out. His review of this thesis was always in such a detail that any kind of mistakes could be spotted, from logic mistakes to grammar errors. He even helped me with practical work and tools like latex, gnuplot, shell scripts and so on. He absolutely provided me with much more than a supervisor usually does. This thesis is dedicated to my supervisor, Dr. Eibe Frank.

Secondly, I would like to thank my project-mate, Nils Weidmann, a lot. I feel so lucky to have worked in the same research area as Nils. He shared with me many great ideas and provided me with much of his work, including datasets, which made my job much more convenient (and made myself lazier). In fact Chapter 6 is the result of joint work with Nils Weidmann. He constructed the Mutagenesis datasets using the MI setting and in an ARFF file format [Witten and Frank, 1999] that allowed me to easily apply the MI methods developed in this thesis to them. He also kindly provided me with the photo files from the online photo library `www.photonewzealand.com` and the resulting MI datasets. Some of the experimental results in Chapter 6 are taken from his work on MI learning [Weidmann, 2003]. But he helped me definitely much more than that. When I met any diffi-

culties during the writeup of my thesis, Nils was usually the first one I asked for help.

Thirdly, many thanks to Dr. Yong Wang. It was him who first introduced me to the "empirical Bayes" methods. He also spent so much precious time selflessly sharing with me his statistical knowledge and ideas, which inspired and benefited my study and research a lot. I am really grateful for this great help.

As a matter of fact, the Machine Learning (ML) family at the Computer Science Department here at the University of Waikato provided me with such a superb environment for study and research. More precisely, I am grateful to: our group leader Prof. Geoff Holmes, Prof. Ian Witten, Dr. Bernhard Pfahringer, Dr. Mark Hall, Gabi Schmidberger, Richard Kirkby (especially for helping me survive with WEKA). I was once trying to name everybody working in the ML lab when I kicked off my project — at that moment we had only three people in the lab, Gabi, Richard and myself. Now there are so many people in the lab, which makes me eventually give up my attempt. But anyway, I would like to thank every one in the ML lab for her/his help or concerns regarding my work.

Help also came from outside the Computer Science Department. More specifically, I am thankful for Dr. Bill Bolstard and Dr. Ray Littler of the Statistics Department. Apart from the lectures they provided to me,[2] they also kindly answered lots of my (perhaps stupid) questions about Statistics.

As for the experiment and development environment used in this thesis, I heavily relied on the WEKA workbench [Witten and Frank, 1999] to build the `MI` package. My work would have been impossible without WEKA. As for the datasets, I would like to thank Dr. Alistair Mowat for providing the kiwifruit datasets. I also want to acknowledge the online photo gallery `www.photonewzealand.com` for their (indirect) provision of a photo dataset through Nils.

Finally, thanks to my family for their consistent support of my project and research.

[2] I always regarded the comments from Ray on the weekly ML discussion as lectures to me.

In fact their question of “haven’t you finished your thesis yet?” has always been my motivation to push the progress of the thesis.

My research was funded by a postgraduate study award as part of Marsden Grant 01-UOW-019, from the Royal Society of New Zealand, and I am very grateful for this support.

Contents

Appendices

List of Figures

List of Tables

Chapter 1
Introduction

Multiple instance (MI) learning has been a popular research topic in machine learning since seven years ago when it first appeared in the pioneering work of Dietterich *et al.* [Dietterich, Lathrop and Lozano-Pérez, 1997]. One of the reasons that it attracts so many researches is perhaps the exotic conceptual setting that it presents. Most of machine learning follows the general rationale of "learning by examples" and MI learning is no exception. But unlike the single-instance learning problem, which describes an example using one instance, the MI problem describes an example with multiple instances. However, there is still only one class label for each example. At this stage we try to avoid any special notation or terminology and simply give some rough ideas of the MI learning problem using real-world examples. Whilst some practical applications of MI learning will be discussed in detail in Chapter 6, we briefly describe them here to give some flavor of how MI problems were identified.

1.1 Some Example Problems

The need for multiple-instance learning arises naturally in several practical learning problems, for instance, the drug activity prediction and fruit disease prediction problems.

The drug activity prediction problem was first described in [Dietterich et al., 1997]. The potency of a drug is determined by the degree its molecules bind to the larger, target molecule. It is believed that the binding strength of a drug molecule is largely determined by its *shape* (or conformation). Unfortunately one molecule can have multiple shapes by rotating some of its internal bonds and it is generally unknown which shape(s) determines the binding. However the potency of a specific molecule, which is either active or inactive, can be observed directly based on the past experiences. One can recognize this as a good instance of MI learning. Each molecule is an example, with each of its shapes as one instance inside it. Thus we have to use multiple instances to present an example. Dietterich *et al.* believed that an additional assumption is intuitively reasonable in this problem, that is, if one of the *(sampled)* shapes of a molecule is active, then the whole molecule is active, otherwise it's inactive. We call this assumption the standard "*MI assumption*". Dietterich *et al.* also provided two datasets associated with the drug activity prediction problem, namely, the Musk datasets. The two datasets describe different musk molecules, with some molecules overlapping between them. These two datasets are the only publicly available real-world MI datasets up to now and are the standard benchmark datasets in the MI domain. We use these datasets throughout this thesis.

The fruit disease prediction problem is another natural case for MI learning. When fruits are collected from different orchards, they are normally stored in the warehouse in batches, one batch for one orchard. After some time (say, 3 to 5 months, which is the usual transportation duration), some disease symptoms are found in some batches. Since the disease may be epidemic, often the whole batch of fruits are infected and exhibit similar symptoms. It would be good to predict which batch is disease-prone before shipment based on some non-destructive measures taken for each fruit as soon as it is collected from the orchards. The task is to predict, given a new batch of fruits, whether it will be affected by a certain disease or not (some months later). This is another obvious MI problem. Each batch is an example with every fruit inside it as an instance. In the training data, a batch is labeled disease-prone (or positive) if the symptoms are observable after the shipment, otherwise disease-free (or negative). Here one may also think the standard MI assumption can fit because the disease may originate in only one or few fruits in a batch. However, even if some fruits indeed exhibit some minor symptoms, the majority of fruits in a batch may be resistant to the disease, rendering the symptoms for the whole batch negligible. Hence the batch can be negative even if some fruits are affected to a small degree.

There are also some real-world problems that are not apparently MI problems. However with some proper manipulation, one can model them as MI problems and generate MI datasets for them. The content-based image processing task is one of the most popular ones [Maron, 1998; Maron and Lozano-Pérez, 1998; Zhang, Goldman, Yu and Fritts, 2002], while the stock market prediction task [Maron, 1998; Maron and Lozano-Pérez, 1998] and computer intrusion prediction task [Ruffo, 2001] are also among them.

The key to modeling content-based image processing as an MI problem is that only some parts of an image account for the key words that describe it. Therefore one can view each image as an example and small segments of that image as instances. Some measures are taken to describes the pixels of an image. Since there are various ways of fragmenting an image and at least two ways to measure the pixels (using RGB values or using Luminance-Chrominance values), there could be many configurations of the instances. The class label of an image is whether its content is about a certain concept and the task is to predict the class label given a new image. When the title of an image is a single simple concept like “sunset” or “waterfall”, the MI assumption is believed to be appropriate because only a small part of a positive image really accounts for its title while no parts of a negative image can account for the title. However, for more complicated concepts of contents, it may be necessary to drop this assumption [Weidmann, 2003]. In this thesis, we consider this application and the others in more detail in Chapter 6.

1.2 Motivation and Objectives

During these years, significant research efforts have been put into MI learning and many approaches have been proposed to tackle MI problems. Although some very good results have been reported on the Musk datasets [Dietterich et al., 1997], the data-generating mechanisms in MI problems remain unclear, even for the well-studied Musk datasets. Thus the MI problem itself may need more attention. Moreover, the reason why some methods perform well and the assumptions they are based on are usually not explicitly explained,[1] which leads to some confusion. Due to the scarcity of MI data, convincing tests and comparisons between MI algorithms are not possible. Hence it is not clear what kinds of MI problems a particular method can deal with. Finally, the relationship between different methods has not been studied enough. Recognizing the essence and the connection between the various methods can sometimes inspire new solutions with well-founded theoretic justifications. Therefore we believe that the study of MI learning still requires stronger and clearer theoretic interpretation, as well as more practical and artificial datasets for the purpose of evaluation.

Therefore in order to capture the big picture of MI learning, we set up the following three objectives:

1. To establish a general framework for MI methods
2. To create some new MI methods within this framework that are strongly justified and make their assumptions explicit
3. To perform experiments on a collection of real-world datasets.

We aimed to achieve these objectives to the extent we could. Although some objectives may be too big to fully accomplish, it is hoped that this thesis improves the interpretation and understanding of MI problems.

[1] We also noticed that some methods actually explicitly expressed assumptions that they never used.

1.3 Structure of this Thesis

The rest of the thesis is organized in seven chapters as follows.

Chapter 2 provides some detailed background on MI problems, the standard MI assumption and published solutions. We also introduce our perspective on the problem and a general framework to analyze current MI methods and create new solutions.

Chapter 3 presents a new MI assumption and a generative model according to the framework introduced in Chapter 2. Then we propose a heuristic MI learning method that *wraps* around normal single-instance methods. We also show that it is suitable for practical classification problems.

In Chapter 4, we put forward some more specific generative models. Under these models we formally derive a way to *upgrade* single-instance learners to deal with MI data. The rationale for the upgraded algorithms is analogous to that of the corresponding single-instance learners but based on some MI assumptions. As an example, we upgrade two popular single-instance learning methods, linear logistic regression and AdaBoost [Freund and Schapire, 1996], to tackle MI problems.

In Chapter 5, we develop a two-level distribution approach to tackle MI problems. Again we explicitly state the underlying generative model this approach assumes. It turns out that this method, in one of its simplest forms, can be regarded as an approximate upgrade of the naive Bayes [John and Langley, 1995] method to the MI setting.

The methods developed in Chapters 3, 4 and 5 are all incorporated in the solution framework introduced in Chapter 2.

Chapter 6 elaborates on the three applications mentioned above and the corresponding datasets. It also presents experimental results of the new methods on these datasets.

Chapter 7 describes the implementation of all the algorithms in this thesis in more detail, and also the process that was used for generating the artificial data used throughout this thesis.

Chapter 8 gives a summary and briefly describes future work. This concludes the thesis.

There are some more implementation details in the Appendices. These details are referenced whenever necessary.

Chapter 2

Background

2.1 Multiple Instance Problems

Multiple instance problems first arose in machine learning in a supervised learning context [Dietterich et al., 1997]. Thus we briefly review supervised learning first. This extends naturally to multiple instance learning.

Supervised learning, more specifically classification, involves "a learning scheme that takes a set of classified examples from which it is expected to learn a way of classifying unseen examples" [Witten and Frank, 1999]. Typically supervised learning has a *training* process that takes some examples described by a pre-defined set of attributes (or a feature vector), and class labels (or responses), one for each example. Attributes can be nominal, ordinal, interval and ratio [Witten and Frank, 1999]. The task for training is to infer a relationship, normally represented as a function, from the attributes to the class labels. If the class labels are nominal values, the task is called "classification". It is called "regression" if the "class" is numeric. Here we are only concerned with the classification task. In this thesis, we only consider two-class classification problems. Nonetheless, it is straightforward to apply them to multi-class problems, e.g. using error-correcting output codes [Dietterich and Bakiri, 1995].

MI learning basically adopts the same setting as single-instance supervised learning

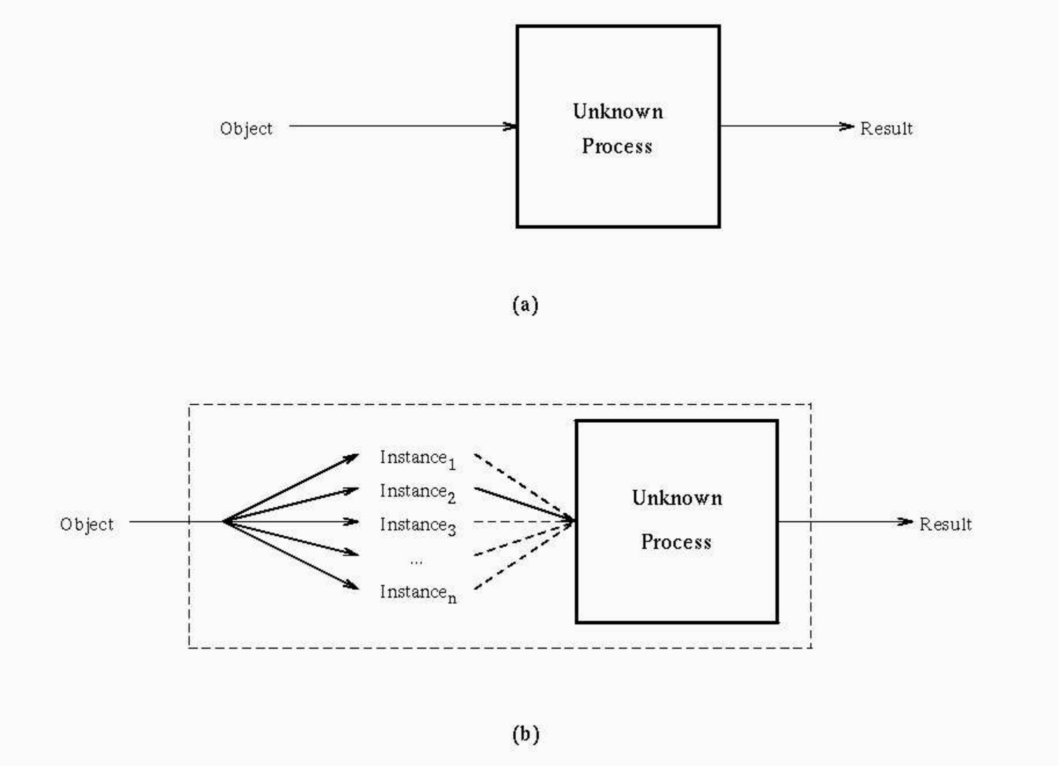


Figure 2.1: Data generation for (a) single-instance learning and (b) multiple-instance learning [Dietterich et al., 1997].

described above. It still has examples with attributes and class labels; one example still has only one class label; and the task is still the inference of the relationship between attributes and class labels. The only difference is that *every one of the examples is represented by more than one feature vector*. If we regard one feature vector as an *instance* of an example, normal supervised learning only has one instance per example while MI learning has multiple instances per example, hence named "*Multiple Instance* learning" problem.

The difference can best be depicted graphically as shown in Figure 2.1 [Dietterich et al., 1997]. In this figure, the "Object" is an example described by some attributes, the "Result" is the class label and the "Unknown Process" is the relationship. Figure 2.1(a) depicts the case in normal supervised learning while in 2.1(b) there are multiple instances in an example. We interpret the "Unknown Process" in Figure 2.1(b) as different from that in Figure 2.1(a) because the input is different. Note that the dashed and solid arrows representing the input of the process in (b) imply that only some of the input instances may be useful. Therefore while the "Unknown Process" in (a) is simply a classification problem, the "Unknown Process" in (b) is commonly viewed as a *two-step* process with a first step consisting of a classification problem and a second step that is a selection process based on the first step and some assumptions.[1] However, as we shall see, there are methods that do not even consider the instance selection step and directly infer the output from the interactions between the input instances. But the assumption of a selection process played such an influential role in the MI learning domain that virtually every paper in this area quoted it. We refer to this standard MI assumption simply as the "MI assumption" throughout this thesis for brevity. Let us briefly review what this assumption entails.

Within a two class context, with class labels {positive, negative}, the "MI assumption" states that an example is positive if at least one of its instance is positive and negative if all of its instances are negative. Note that this assumption is based on instance-level class labels, thus the "Unknown Process" in Figure 2.1(b) has to consist of two steps: the first step provides the instances' class labels and the MI assumption is applied to the second step. However, we noticed that several MI methods do *not* actually follow the MI assumption (even though it is generally mentioned), and do not explicitly state which assumptions they use. As a matter of fact, we believe that this assumption may not be so essential to make accurate predictions. What matters is the *combination* of the model from the first step and the assumption used in the second step. Given a certain type of model for classification, it may be appropriate to explicitly drop the MI assumption and establish other assumptions in the second step. We will discuss this in Section 2.4.

The two-step paradigm itself is only one possibility to model the MI problem. We noticed that in general, when extending the number of instances of an example from one to many, we may have a potentially very large number of possibilities to model the relationship between the set of instances and their class label. For

[1] We may like to call the selection process a "parametric instance selection" process because it is based on the model built in the classification process.

the instances within an example, there may be *ambiguity*, *redundancy*, *interactions* and many more properties to exploit. The two-step model and the MI assumption may be suitable to exploit ambiguity [Maron, 1998] but if we are interested in other properties of the examples, other models and assumptions may be more convenient and appropriate. In practice, we may need strong background knowledge to choose the right way to model the problem a priori. However in most cases we actually lack such knowledge. Consequently it is necessary that we try a variety of models or assumptions in order to come up with an accurate representation of the data.

2.2 Current Solutions

As mentioned above, there are a variety of methods that have been developed to tackle MI problems. Almost all of them are special-purpose algorithms that are either created for the MI setting or upgraded from the normal single-instance learners. Unfortunately some of them do not provide sufficient explanation of the underlying assumptions or generative models. Hence the working mechanisms of the methods remain unclear. The algorithms are discussed in a chronological order in the following section and comments are provided whenever possible. We discuss the algorithms developed before 2000 and those after 2000 separately. We did so because we observed that the post-2000 methods are based on different assumptions than the pre-2000 methods.

2.2.1 1997-2000

The first MI algorithm stems from the pioneering paper by Dietterich *et al.* [1997], which also introduced the aforementioned Musk datasets. The APR algorithms [Dietterich et al., 1997] modeled the MI problem as a two-step process: a classification process that is applied to every instance and then a selection process based on the MI assumption. A single Axis-Parallel hyper-Rectangle (APR) is used as the pattern to be found in the classification process. As a parametric approach,[2] the objective of these methods is to find the parameters that, together with the MI assumption, can best explain the class labels of all the examples in the training data. In other words, they look for parameters involved in the first step according to the observed class label formed in the second step and the assumed mechanism between the two steps. There are standard algorithms that can build an APR (i.e. a single if-then rule). Unfortunately they do not take the MI assumption into account. Thus a special-purpose algorithm is needed. Several basic APR algorithms were proposed [Dietterich et al., 1997], but interestingly the best method for the Musk data was not among them. The best APR algorithm for the Musk data consists of an iterative process with two steps: the first step expands an APR from a positive "seed" instance using a back-fitting algorithm, and the second step selects useful features greedily based on some specifically designed measures. It turned out that the APR that best explain the training data does not generalize very well. Hence the objective was changed a little bit to produce the lowest generalization error, and the kernel density estimation (KDE) was used to refine some of the parameters — the boundaries of the hyper-rectangle. These tuning steps resulted in an algorithm called "iterated-discrim APR", that gave good results on the Musk datasets.

It is interesting that Dietterich *et al.* based all the algorithms on the "APR pattern and the MI assumption" combination and never considered to drop either of them even when difficulties were encountered. This might be due to their background knowledge that made them believe an APR and the MI assumption are appropriate for the Musk data. However, using KDE may have already introduced a bias into the parameter estimates, which indicates that it might be more appropriate to model the Musk datasets in a different way.

In spite of the above observation, the "APR and the MI assumption" combination dominated the early stage of MI learning. Researchers from the computational learning theory played an active role in MI learning before 2000. Some PAC al-

[2] In this case, the parameters to be found are the useful features and the bounds of the hyper-rectangle along these features.

gorithms were developed to look for the APR that Dietterich *et al.* defined [Long and Tan, 1998; Blum and Kalai, 1998]. They also proved that finding such an APR under the MI assumption is actually an NP-complete problem. Some more practical MI algorithms were also developed in this domain, such as MULTINST [Auer, 1997]. PAC learning theory tends to view classification as a deterministic process. Instances with the class labels that violate the assumed concept are usually regarded as "noisy". Indeed, coming up with the idea of varying noise rates for the examples and assuming a specific distribution of the number of instances per example, Auer [1997] ended up estimating the examples' misclassification errors and looking for an APR to minimize this estimated error. Note that since MULTINST tries to calculate the expected number of instances per bag that fall into the hypothesis for the positive class (in this case an APR), it adheres closely to the MI assumption.

Another way to view the classification problem is from a probabilistic point of view, which is prevalently adopted by statisticians and the researchers in the statistical learning domain [Hastie, Tibshirani and Friedman, 2001; Vapnik, 2000; McLachlan, 1992; Devroye, Györfi and Lugosi, 1996]. Normally one thinks of a joint distribution over the feature vector variable X and the class variable Y. By calculating the marginal probability of Y conditional on X, $Pr(Y|X)$, we can predict the probability of the class label of a test feature vector. According to statistical decision theory, one should make the prediction based on whether the probability is over 0.5 in a two-class case.

But this is in single-instance supervised learning. In the MI domain, the first (and so far the only published) probabilistic model is the Diverse Density (DD) model proposed by Maron [Maron, 1998; Maron and Lozano-Pérez, 1998]. As will be described in more details in Chapter 4 and Chapter 7, DD was also heavily influenced by the "APR and the MI assumption" combination and the two-step process. The DD method actually used the maximum binomial log-likelihood method, a statistical paradigm used by many normal single-instance learners like logistic regression, to search for the parameters in the first step. In particular, to model an APR in the first step, it used a radial form or a "Gaussian-like" form to model $Pr(Y|X)$.

Because of the radial form, the pattern (or decision boundary) of the classification step is an Axis-Parallel hyper-Ellipse (APE) instead of a hyper-rectangle.[3] In the second step, where we assume that we have already obtained each instance's class probability, we still need ways to model the process that decides the class label of an example.[4] There were two ways in DD, namely the noisy-or model and the most-likely-cause model, that are both probabilistic ways to model the MI assumption.

In the single-instance case, the process to decide the class label of each example (i.e. each instance) can be regarded as a one-stage Bernoulli process. Now, since we have multiple instances in an example, it seems natural to extend it to a multiple-stage Bernoulli process, with each instance's (latent) class label determined by its class probability in one stage.[5] And this is exactly the "noisy-or" generative model in DD. As we shall see, according to [Maritz and Lwin, 1989], a very similar way of modeling was adopted by some statisticians in as early as 1943 [von Mises, 1943]. However we can also model the process as an one-stage Bernoulli process if we assume some way to "squeeze" the multiple probabilities (one per instance) into one probability. The most-likely-cause model in DD is of this kind. Either way, we can form a binomial log-likelihood function, either multi-stage or one-stage. The noisy-or model computes the probability of seeing all the stages negative and the complement, the probability of seeing at least one stage positive. The most-likely-cause model picks only one instance's probability in an example to form the binomial log-likelihood. It selects the instance within an example that has the highest probability to be positive. Both processes to generate a bag's class label are model-based[6] and are compatible with the MI assumption. By maximizing the log-likelihood function we can find the parameters involved in the radial formulation of $Pr(Y|X)$. This is how DD "recovers" the instance-level class probability, $Pr(Y, X)$. With the virtues of the maximum likelihood (ML) method, namely its consistency and efficiency, one can usually assure the correctness of the solution if

[3] Note that the function for a rectangle is not differentiable whereas that of an ellipse is. But in the sense of classification decision making, they are very similar.

[4] The process to decide an example's class label was called "generative model" in DD.

[5] Note that the normal binomial distribution formula $C_n^r p^r(1-p)^{n-r}$ does not apply here because in this Bernoulli process, the probability p changes from stage to stage.

[6] The processes are based on the radial formulation of $Pr(Y|X)$.

the underlying assumptions and generative model (described in [Maron, 1998]) are true. In the testing phase, the estimated parameters and the same Bernoulli process used in training provide a class probability for a test example. Once again we decide its class label using the probability threshold of 0.5.

Within the most-likely-cause model, the log-likelihood function involves the $max(.)$ functions and becomes non-differentiable, making it hard to maximize the log-likelihood function. That is why the EM-DD was proposed [Zhang and Goldman, 2002] some years later. EM-DD uses the EM algorithm [Dempster, Laird and Rubin, 1977] to overcome the non-differentiability in the optimization of the log-likelihood function. Thus in methodology, EM-DD is simply DD.[7] Nonetheless, it can be shown (see Appendix D) by some theoretical analysis and an illustrative counter-example that such an attempt is generally a failure. The theoretic proof of the convergence of EM-DD is also problematic. As a result, EM-DD will not generally find a maximum likelihood estimate (MLE) of the parameters. Since DD is an ML method, EM-DD's solution will not be a correct one if it cannot find the MLE. EM-DD was found to produce a very good result on the Musk data but this was due to a flawed evaluation involving intensive parameter tuning on the test data. Not surprisingly, the EM-DD algorithm did not work better than DD on the content-based image retrieval task [Zhang, Goldman, Yu and Fritts, 2002].

2.2.2 2000-Now

The new millennium saw the breaking of the MI assumption and abandonment of APR-like formulations. Virtually no new methods (apart from a neural network-based method) created since then use the MI assumption, although interestingly enough some of them were motivated based on this assumption. Hence it may not be fair to compare these methods with the methods developed before 2000 because they were based on different assumptions. However, one can usually compare them in the sense of verifying which assumptions and models are more appropriate for

[7] That is why we list it here and not among the new methods after 2000, although it was published in 2002.

the real-world data.

The new methods created since 2000 were all aimed to upgrade the single-instance learners to deal with MI data. The methods upgraded so far are decision trees [Ruffo, 2001], nearest neighbour [Wang and Zucker, 2000], neural networks [Ramon and Raedt, 2000], decision rules [Chevaleyre and Zucker, 2001] and support vector machines (SVM) [Gärtner, Flach, Kowalczyk and Smola, 2002]. Nevertheless the techniques involved in some of these methods are significantly different from those used in their single-instance parents. We can categorize them into "instance-based" methods and "metadata-based" methods. The term "instance-based" denotes that a method is trying to select some (or all) representative instances from an example and model these representatives for the example. The selection could be based on the MI assumption or, more often after 2000, not. The term "metadata-based" means that a method actually ignores the instances within an example. Instead it extracts some meta-data from an example that is no longer related to any specific instances. The metadata-based approaches *cannot* possibly adhere to the MI assumption because the MI assumption must be associated with instance selection within an example. We briefly describe the post-2000 methods using this categorization, which is also the backbone of the framework we will discuss in the next section.

The instance-based approaches are the nearest neighbour technique, the neural network, the decision rule learner, and the SVM (based on a multi-instance kernel).

The MI nearest neighbour algorithms [Wang and Zucker, 2000] introduces a measure that gives the distance between two example, namely the Hausdorff distance. It basically regards the distance between two examples as the distance between the representatives within each example (one representative instance per example). The selection of the representative is based on the maximum or minimum of the distances between all the instances from the two examples. While it is not totally clear from the paper [Wang and Zucker, 2000] what the so-called "Bayesian-KNN" does in the testing phase, the "Citation-KNN" method definitely violates the MI assumption because it decides a test example's class label by the majority class of its nearest examples. Thus in general it does not classify an example based on whether at least one of its instance is positive or all of the instances are negative.

On the other hand, MI neural networks [Ramon and Raedt, 2000] closely adhered to the MI assumption. They adopt the same two-step framework used in the APR method [Dietterich et al., 1997] as described above. As a matter of fact, one may recognize that searching for parameters in the aforementioned two-step process is well suited for a two-level neural network architecture. Neural network is used to learn a pattern in the classification step, and a model-based instance selection method is applied in the second step. In the first step the family of patterns is not explicitly specified but implicitly defined according to complexity of the network constructed. In the second step, like the most-likely-cause model in DD [Maron, 1998], the neural network picks up the instance with the highest output value in an example.[8] Backpropagation is used to search for the parameter values. Therefore it can be said that this method is based on the MI assumption. Indeed, the reported results obtained seemed to be very similar to those of the DD algorithm on the Musk datasets.

NaiveRipperMI [Chevaleyre and Zucker, 2001] is a modification of the rule learner RIPPER [Cohen, 1995] with a different counting method. Instead of counting how many instances are covered by a hypothesis, it counts how many examples are covered. If at least one instance of an example is covered, the whole example is counted. Because positive and negative examples are treated the same way, this violates the MI assumption. In fact this method could find the hyper-rectangle that covers all negative examples but no positive ones. Since NaiveRipperMI [Chevaleyre and Zucker, 2001] emphasized so much on the MI assumption, we assume[9] that it does what the MI assumption states in the testing procedure. However, this would mean that it is not consistent with what happens in the training procedure.

The SVM with the MI kernel [Gärtner et al., 2002] also violates the MI assumption. The MI kernel simply replaces the standard dot product by the sum over all pairwise

[8] Since the output value is $\in [0, 1]$, we can regard it as the probability to be positive.

[9] There is no information on how a test example is classified in [Chevaleyre and Zucker, 2001].

dot products between instances from two examples. This can be combined with another non-linear kernel, e.g. RBF kernel. This effectively assumes that the class label of an example is the true class label of all the instances inside it, and attempts to search for the hyperplane that can separate all (or most of)[10] the training examples in an extended feature space (because of the RBF kernel function). Since this is done the same way for both positive and negative examples, the MI assumption is not used in this method at all. Indeed, we observe that some methods, including SVMs, that do not model the probability $Pr(Y|X)$ directly, will find the MI assumption very hard to apply, if not impossible. It would be very convenient for those methods to have other assumptions associated with the measure they attempt to estimate.

The metadata-based approach is implemented in the MI decision tree learner RELIC and the SVM based on a polynomial minimax kernel. This approach extracts some metadata from each example, and regards such metadata as the characteristics of the examples. When a new example is seen, we can directly predict its class label with regards to the metadata without knowing the class labels of the instances. Therefore each instance is not important in this approach. What matters is the underlying properties of the instances. Hence we cannot tell which instance is positive or negative because an example's class label is associated with the properties that are presented by the attribute values of all the instances. Hence this approach cannot possibly use the MI assumption.

The MI decision tree learner RELIC [Ruffo, 2001] is of this kind. In each node in the tree, RELIC partitions the examples according to the following method:

- For a nominal attribute with R values, say θ_r where $r = 1, 2, \ldots R$, it assigns an example to the r^{th} subset if there is at least one instance in the example whose value of this attribute is θ_r.
- For a numeric attribute, and given a threshold θ, there will be two subsets to be chosen: the subset less than θ and that greater than it. It assigns an example in either subset based on two types of tests. The first type assigns

[10] The regularization parameter C in SVM will tolerate some errors in the training data.

an example into the subset less than θ if the minimum of this attribute values of all the instances within the example is less than or equal to θ and otherwise to the subset greater than θ. For the second type, it assigns an example into the subset less than θ if the maximum of this attribute values of all the instances within the example is less than or equal to θ and otherwise to the subset greater than θ.

- Then it seeks the best θ according the entropy measure, same way as the single-instance tree learner C4.5 [Quinlan, 1993]. Note that for numeric attributes, it looks for the best of the two types of tests simultaneously so that only one type of tests and one θ will be selected for each numeric attribute.

The way that RELIC assigns examples to subsets means that it is equivalent to extracting some metadata, namely minimax values, from each example and applying the single-instance learner C4.5 to the transformed data. Since RELIC examines the attribute values along each dimension individually, such metadata of an example does not correspond to any specific instance inside it, although it is possible, but very unlikely, to match the instances to the minimax values of all the attributes simultaneously. Moreover, in the testing phase, we can directly tell an example's class label using the tree without getting the class labels of the instances. Thus the MI assumption obviously does not apply.

The SVM with a polynomial minimax kernel explicitly transform the original feature space to a new feature space where there are twice the number of attributes as in the original one. For each attribute in the original feature space, two new attributes are created in the transformed space: “minimal value” and “maximal value”. It then maps each example in the original feature space into an instance in the new space by finding the minimum and maximum value of each attribute for the instances in that example. Clearly some information is lost during the transformation and this is significantly different from the two-step paradigm used in [Dietterich et al., 1997]. The MI assumption cannot possibly apply. Note this is effectively the same as what RELIC does.

The assumption of the metadata approach is that the classification of the examples is only related to the metadata (in this case the minimax values) of the examples, and that the transformation does not lose (or lose little) information in terms of classification. The convenience of the approach is that it transforms the multi-instance problem to the common mono-instance one. In general, the metadata approach enables us to transform the original feature space to other feature spaces that facilitate the single-instance learning. The other feature spaces are not necessarily the result of simple metadata extracted from the examples. They could be, for instance, a model space where we build a model (either a classification model or a clustering model) for examples and transform each example into one instance according to, say, the count of its instances that can be explained by the model. Methods similar to this are actively being researched [Weidmann, 2003]. The validity of such transformations really depends on the background knowledge. If one believes that interactions or relationships between instances account for classification of an example, then this approach may outperform methods that do not have such a sophisticated view of the problem. In this thesis, we refer to all the methods that transform the original feature space to another feature space as the "metadata-based approach", no matter how complicated the extracted metadata may be.

In summary, the methods developed in the earlier stage of MI learning usually have an APR-like formulation and hold the MI assumption whereas the methods developed later on often implicitly drop the MI assumption and are based on other types of models.

2.3 A New Framework

As mentioned in the previous section, we can categorize all the current MI solutions into a simple framework. As for the methods before 2000, it is quite obvious that they belong to the "instance-based approaches" because they strictly adhered to the MI assumption, which implies that one implicitly assumes a class label for each instance. We now present a hierarchical framework giving our view of MI learning.

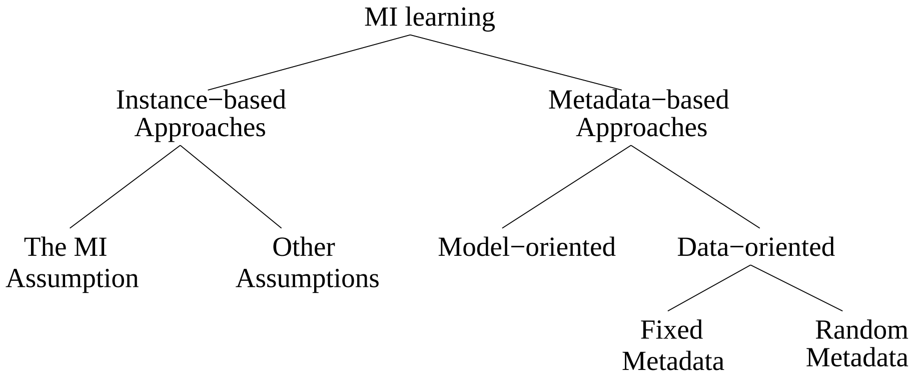


Figure 2.2: A framework for MI Learning.

This is shown in Figure 2.2.

It is now almost a convention that the MI assumption is associated with MI learning. In this thesis, we generalize the definition of multiple instance learning and allow the algorithm developers to plug in whatever assumption they believe reasonable. Thus "MI learning" in our framework is based on generalizing the MI assumption. As already discussed we categorize MI methods into two categories: instance-base approaches and metadata-based approaches. We have already analyzed the current solutions based on this distinction, and we will create more methods in this thesis that belong to either one of the two categories. We can specialize the two categories further.

In the instance-based approach, one normally estimates some parameters of a function mapping the feature variable X to the class variable Y. This function is then used to form a prediction at the bag level. All current instance-based methods for MI learning amount to estimating the parameters of the function that enable them to predict the class label of unseen examples. Some of these methods are based on the MI assumption but some are not, which results in two sub-categories. We will also develop some more methods within the sub-category that *are not based on the MI assumption*. We will explicitly state our assumptions and present the generative models that the methods are based on.

The metadata-based approach has already been discussed in the last section. The metadata could either be directly extracted from the data, so-called "data-oriented" in the framework shown in Figure 2.2, or from some models built on the data, called "model-oriented".

The two-level classification method [Weidmann, Frank and Pfahringer, 2003; Weidmann, 2003] is the only published approach that is "model-oriented". It builds a first-level model using a single-instance (either supervised or unsupervised) learner to describe the potential patterns in the whole instance space as the metadata. It then applies a second-level learner to determine the model based on the extracted patterns. A second-level learner is a single-instance (supervised) learner. Thus it effectively transforms the original instance space into a new ("model-oriented") instance space that single-instance learners can be applied to.

For example, we can apply a clustering algorithm — at the first level — to the original instances and build a clustering model from the (original) instance space. Then we can construct a new single-instance dataset, with every attribute corresponding to one cluster extracted, and each instance corresponding to one example in the original data. The new instances' attribute values are the number of instances of the corresponding example that fall into one cluster. Finally we apply another single-instance learner, say a decision tree, to the new data to build a second-level model. At testing time, the first-level model is applied to the test example to extract the metadata (and generate a new instance), and the second-level learner to classify the test example according to the model built on the (training) metadata. Of course there are many combinations of the first and second-level learners, and they are not further described in this thesis. Interested readers should refer to [Weidmann, 2003] for more detail.

In the "data-oriented" sub-category, we can further specialize into "fixed metadata" and "random metadata" sub-categories. All of the current metadata-based methods (i.e. RELIC and SVM based on a minimax kernel) are data-oriented. In addition, the metadata extracted is thought to be fixed and directly used to find the function mapping the metadata to Y. However we can regard the metadata as random and governed by some distributions, which results in another sub-category. Here we assume the data within each example is random and follows a certain distribution. Thus we can extract some low-order sufficient statistics to summarize the data and regard the statistics as the metadata. In this case, the metadata (statistics) have a sampling distribution parameterized by some parameters. If we further think of these parameters as governed by some distribution, the metadata is necessarily random, not only wandering around the parameters within a bag but from bag to bag as well. This is the thinking behind our new two-level distribution approach discussed in Chapter 5. It turns out that when assuming independence between attributes, it constitutes an *approximate* way to upgrade the naive Bayes method to deal with MI data, which has not been tried in the MI learning domain. We also discovered the relationship between this method and the empirical Bayes methods in Statistics [Maritz and Lwin, 1989].

2.4 Methodology

When we generalize the assumptions and approaches of MI learning, we find much flexibility within our framework described above. Nonetheless, we still like to restrict our methods to some scope so that we may easily find theoretical justifications for them. We propose that it would be desirable for an MI algorithm to have the following three properties:

1. The assumptions and generative models that the method is based on are clearly explained. Because of the richness of the MI setting, there could be many (essentially infinitely many) mechanisms that generate MI data. It is very unlikely that one method can deal with all of them. A method has to be based on some assumptions. Thus it is important to state the assumptions the method is based on and their feasibility.

2. The method should be consistent in training and testing. Note “consistency” here means that a method should make a prediction for a new example in

the same way as it builds a model on the training data.[11] For example, if a method tries to build a model at the instance level and is based on a certain assumption other than the MI assumption, then it should also predict the class label of a test example using that assumption.

3. When the number of instances within each example reduces to one, the MI method degenerates into one of the popular single-instance learning algorithms. Although this is not such an important property compared to the former two, it is useful when we upgrade a single-instance learner, which is theoretically well founded, to deal with MI data.

Even though not all current MI methods hold the above three properties, we aim to achieve them in this project. In order to do so we develop the following methodology for this thesis.

1. We explicitly drop the MI assumption but state the corresponding new assumptions whenever new methods are created. The underlying generative model of each new method will be explicitly provided so that one may clearly understand what kind of problem the methods can solve.

2. We adopt a statistical decision theoretic point of view, that is, we always assume a joint probability distribution over the feature variable X (or other new variables introduced for MI learning) and the class variable Y, $Pr(X,Y)$, as assumed by most of single-instance learning algorithms. We base all our modeling purely on this distribution. Although we can end up with different methods by factorizing the joint distribution differently, the root of them is the same.

3. As the standard single-instance learners are mostly well justified and empirically verified, we are interested in creating new methods related to them. Even if we develop a new method that is in totally different context, we try to show the relationship between this method and some single-instance learning algorithms whenever possible.

[11] This is different from the notation of consistency in a statistical context.

2.5 Some Notation and Terminology

To avoid confusion, this section explicitly lists the common notation that will be used in this thesis, and some terminology. Special notation and terminology will be used together with proper explanations.

An “example” in the MI domain is also called a “bag” or an “exemplar” and we will use all three terms in this thesis. Likewise an “instance” is sometimes called a “feature vector” or a ”point” in the feature space. Every instance is regarded as a value of the “feature variable” X whereas its class label is a value of the class variable Y. Y is also called the “response variable” and “group variable”. An “attribute” will also be called a “feature” or a “dimension”. There are many names for the algorithms in normal single-instance (or mono-instance) supervised learning like “propositional learner”, or “Attribute-Value (AV) learner”. We will sometimes use them without distinction.

There is also some notation related to the joint probability distribution $Pr(X, Y)$. In classification problems we are more concerned with the conditional (or marginal) probability of Y, $Pr(Y|X)$. We also call it the “posterior probability” and/or the point-conditional probability because it is conditional on a certain point x. But we also have the marginal probability of X, $Pr(X|Y)$ which we also call the group-conditional probability. Of course we call $Pr(X)$ and $Pr(Y)$ the “prior probabilities” of X and Y respectively. Note when X is numeric, we abuse the symbol “$Pr(.)$” because it is a density function that we refer to. However we will not distinguish this difference in the notation.

Chapter 3
A Heuristic Solution for Multiple Instance Problems

This chapter introduces new assumptions for MI learning, and we discard the standard MI assumption. We regard the class label of a bag as a property that is related to all the instances within that bag. We call this new assumption "the collective assumption" because the class label of a bag is now a collective property of all the corresponding instances. Why can the collective assumption be reasonable for some practical problems like drug activity prediction? Because the features variables (in this case measuring the conformations, or shapes, of a molecule) usually cannot absolutely explain the response variables (a molecule's activity), it is appropriate to model a probabilistic mechanism to decide the class label of a data point in the instance space. The collective assumption means that the probabilistic mechanisms of the instances within a bag are intrinsically related, although the relationship is unknown to us. Consider the drug activity prediction problem: a molecule's conformations are not arbitrary in the instance space but confined to some certain areas. Thus if we assume that the mechanism determining the class label of a molecule is similar to the mechanism determining the (latent) class labels of all (or most) of the molecule's conformations, we may better explain the molecule's activity. Even if the activity of a molecule were truly determined by only one specific shape (or a very limited number of shapes) in the instance space, it would have a very small probability of being sampled. Together with some measurement errors, the samples of a molecule are very likely to wander around the "true" shape(s). Therefore it is more robust to model the collective class properties of the instances within a bag rather than that of some specific instance. We believe this is true for many practical MI datasets, including the musk drug activity datasets.

The "collective assumption" is a broad concept and there is great flexibility under this assumption. In Section 3.1 we present several possible options to model exact generative models based on this assumption. We further illustrate it with an artificial dataset generated by one exact generative model in Section 3.2. This generative model is strongly related to those used in Chapters 4 and 5. In fact, all the methods developed in this thesis are based on some form of the collective assumption. Section 3.3 presents a heuristic wrapper method for MI learning that is based on the collective assumption [Frank and Xu, 2003]. It will be shown that in some cases it can perform classification pretty well even though it introduces some bias into the probability estimation. We interpret and analyze some properties of the heuristic in Section 3.4. Note that some of the material in this chapter has been published in [Frank and Xu, 2003].

3.1 Assumptions

Under the collective assumption, we have several options to build an exact generative model. We have to decide which options to take in order to generate a specific model. We found that answering the following question is helpful in making the decision:

1. **How to define the class label property of an instance and of a bag?**
   Suppose we use a function of the class variable Y, $C(Y|.)$ to denote the class label property, then what is the exact form of $C(Y|.)$? In single-instance statistical learning, we usually model $C(Y|X)$ to be related to the posterior probability function $Pr(Y|X)$: we either use $Pr(Y|X)$ itself or its logit

transform, the log-odds function $\log \frac{Pr(Y=1|X)}{Pr(Y=0|X)}$. In MI learning, we can also model it at the bag level, i.e. we can build a function of $C(Y|B)$ where B are the bags. Thus we can also have $Pr(Y|B) = Pr(Y|X_1, X_2, \cdots, X_n)$ and $\log \frac{Pr(Y=1|B)}{Pr(Y=0|B)} = \log \frac{Pr(Y=1|X_1,X_2,\cdots,X_n)}{Pr(Y=0|X_1,X_2,\cdots,X_n)}$ where a bag $B = b$ has n instances $x_1, x_2, \cdots, x_n$. Note that in this thesis we restrict ourselves to the same form of the property for instances and bags. For example, if we model $Pr(Y|X)$ for the instances, we also model $Pr(Y|B)$ for bags instead of the log-odds. We will show the reason for doing so in the answer to the next question.

2. **How to view the members of a bag and what is the relationship between their class label properties and that of the bag?**

   Almost all the current MI methods regard the members (instances) of a bag as a finite number of fixed and unrelated elements. However we have a different point of view. We think of each bag as a population, which is continuous and generate instances in the instance space in a dense manner. What we have in the data for each bag are some samples randomly sampled from its population. This point of view actually relates all the instances to each other *given a bag*, that is, they are all dominated by the specific distribution of the population of that bag. Every bag is unbounded, i.e., it ranges over the whole instance space. However its distribution may be bounded, i.e. its instances may only be possibly located in a small region of the instance space.[1] If one really thinks of the bags' elements as random selected from the whole instance space, we can still fit this into our thinking by modeling each bag as with a uniform distribution. Note that the distributions of different bags are different from each other and bags may overlap. Thus each point in the instance space x (that is, an instance) will have different density given different bags. Therefore, unlike in normal single-instance learning that assumes $Pr(X)$, we have $Pr(X|B)$ instead. We still regard each point in the instance space as having its own class label (that could be determined by either a deterministic or a probabilistic process) but this is *unobservable* in the data. What is observable is the class label of a bag that is determined using the instances'

[1] In fact if the bags are bounded, we can always think of their distributions as bounded ones.

(latent) class label properties.

Now how to decide the class label of a bag given those of its instances? There are two options here: the population version and the sample version. Since we take the above perspective for a bag and we have already defined the class label property $C(Y|.)$, one application of the collective assumption is to take the *expected* property of the population of each bag as the class property of that bag. Given a bag b, we calculate

$$C(Y|b) = E_{X|b}[C(Y|X)] = \int_X C(Y|x)Pr(x|b)\,\mathrm{d}x \tag{3.1}$$

We simply regard the bags' class label property as the *conditional expectation* of the class property of all the instances given b. This is the population version for determining the class label of a bag $C(Y|b)$. It is not related to any individual instance, but we must know $Pr(x|b)$ exactly to calculate the integral. However, there is also a sample version of $C(Y|b)$. If given b we can sample n_b instances from the instance space (in other words, there are n_b instances in b), then $C(Y|b) = \frac{1}{n_b}\sum_{i=1}^{n_b} C(Y|x_i)$. Since instances are drawn via random sampling, we can give each instance an equal weight and calculate the weighted average no matter what the distribution $Pr(x|b)$ is. The population and the sample version are approximately the same when the in-bag sample size is large, but there may be large difference if the sample size is small. For example, if only one instance is sampled per bag. In this case the sample version reduces to the single-instance case because an instance's (in this case also a bag's) class label is determined by its own class label property. However, the population version will still determine the instance's class label by the overall class property of its bag. Consequently it may be less desirable because it does not degenerate naturally to the single-instance case.

It is now clear why we choose consistent formulations of $C(Y|.)$ for both bags and instances — because we regard $C(Y|B)$ simply as the expected value of $C(Y|X)$ conditional on the existence of a bag $B = b$. While there may be other applications of the collective assumption, in this thesis we take this perspective only. There are still two options, as discussed above, for generating MI data under the collective assumption: the sample version and the population version. In this thesis we use the sample version because of its elegant degradation into single-instance supervised learning.

3. **How to model** $Pr(X|B)$ **?**

   Whether using the population or the sample version, we need to know the conditional density of the instances, $Pr(X|B)$ because we need to generate the instances of each bag from this distribution. There are also two possibilities to model $Pr(X|B)$: one is to model it indirectly using $Pr(X)$ and the other is to model $Pr(X|B)$ directly.

   The first option still assumes the existence of $Pr(X)$, thus we are now in the same framework as normal single-instance learning where we assume both $Pr(X)$ and $Pr(Y|X)$. What is new is that we introduce a new variable B to denote the bags and define $Pr(X|B)$ in terms of $Pr(X)$. In particular, we assume the distribution of each bag $Pr(X|B)$ is bounded and only occupies a limited area in the instance space. As a result we model the *conditional density* of the feature variable X given a bag $B = b$ as

$$Pr(x|b) = \begin{cases} \frac{Pr(x)}{\int_{x \in b} Pr(x)\ \mathrm{d}x} & \text{if } x \in b, \\ 0 & \text{otherwise}. \end{cases} \tag{3.2}$$

   Note that we abuse the notation $Pr(.)$ here because we really have a density function of X instead of probability if X is numeric. To put it another way, the distribution of each bag is simply the normalized instance distribution $Pr(X)$, restricted to the corresponding range. In both the population and the sample version of the generative model, we need Equation 3.2 to generate instances for one bag. In the population version we also need it to create the class label of a bag b, whereas in the sample version we do not need it because the class label of a bag is not related to a specific form of the density function of its instances.

   The second option for modeling $Pr(X|B)$ does not assume the existence of

$Pr(X)$. Indeed if all we need is $Pr(X|B)$, why should we still rely on the single-instance statistical learning paradigm? Given a bag b, we can directly model $Pr(x|b)$ as some distribution, say a Gaussian, parameterized by some parameters. Now a bag is basically described by its parameters because once the parameters of a bag are decided, that bag has been formed. Thus we need some mechanism to generate the parameters of the bags — we can conveniently regard the parameters themselves as distributed according to some "hyper-distribution". The data generation process is exactly the same as in the first option, for both the population version and the sample version. Note that if we model $Pr(X|B)$ directly, $Pr(X)$ may or may not exist, depending on the specific distribution involved.

The above two options may coincide sometimes, as will be shown in Section 3.2, but in general they generate different data. Note that although the conditional density function $Pr(X|B)$ must be specified in order to generate instances for each bag, it is not important for the instance-based learning algorithms that will be discussed (particularly in Chapter 4) because they are based on the sample version of the generative model, in which $Pr(X|B)$ is not relevant to the class probability of the bags $Pr(Y|B)$. However, in Chapter 5, we pay much attention to $Pr(X|B)$ as we develop a method that models it in a group-conditional manner (i.e. conditional on Y).

Now that we have answered the above questions, we are able to specify how to generate a bag of instances and how to generate the class label of that bag. Thus it is the time to generate an MI dataset based on an exact generative model, under the collective assumption. In the following section, we illustrate the above specifications via an artificial dataset, specifying the answers of the above question as follows. First, the class label property $C(Y)$ is the posterior probability at both the instance and the bag level. In other words, we model $Pr(Y|B)$ and $Pr(Y|X)$. Second, we think of each bag as a hyper-rectangle in the instance space and assume the center (i.e., the middle-point of the hyper-rectangle) of each bag is uniformly distributed. Thus the bags are bounded and the instances for each bag are drawn from within

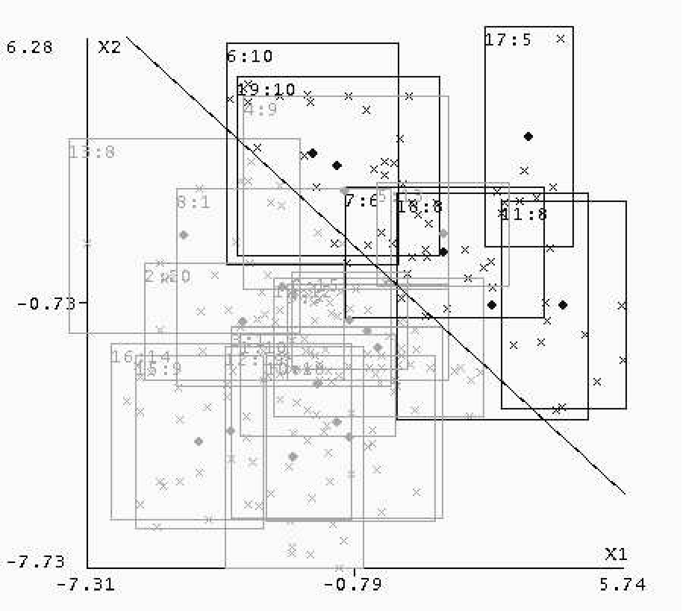


Figure 3.1: An artificial dataset with 20 bags.

the corresponding hyper-rectangle. We model $Pr(Y|X)$ as a linear logistic model, i.e. $Pr(Y=1|X) = \frac{1}{1+\exp(-\beta X)}$, and based on the sample version of the generative model, $Pr(Y|B) = \frac{1}{n}\sum_{i=1}^{n} Pr(Y|X_i)$, where n is the number of instances in a bag. Finally we take the first option for the last question and assume the existence of $Pr(X)$, which we model as a uniform distribution. Thus the conditional density in Equation 3.2 is simply a uniform distribution within the region that a bag occupies.

3.2 An Artificial Example Domain

In this section, we consider an artificial domain with two attributes. More specifically, we created bags of instances by defining rectangular regions and sampling instances from within each region. First, we generated coordinates for the centroids of the rectangles according to a uniform distribution with a range of $[-5, 5]$ for each of the two dimensions. The size of a rectangle in each dimension was chosen from 2 to 6 with equal probability (i.e. following a uniform distribution). Each rectangle was used to create a bag of instances. To this end we sampled n instances from within a rectangle according to a uniform distribution. The value of n was chosen from 1 to 20 with equal probability. Note that although we adopt the first option to model $Pr(X|B)$ from the previous section, it is identical to the second option if we assume a different uniform distribution for each bag, and that the parameters of the uniform distributions are dominated by another (hyper-) uniform distribution.

It remains the question how to generate the class label for a bag. As mentioned before, our generative model assumes that the class probability of a bag is the average class probability of the instances within it, and this is what we used to generate the class labels for the bags. The instance-level class probability was defined by the following linear logistic model:

$$Pr(y = 1|x_1, x_2) = \frac{1}{1 + e^{-3x_1 - 3x_2}}$$

Figure 3.1 shows a dataset with 20 bags that was generated according to this model. The black line in the middle is the instance-level decision boundary (i.e. where $Pr(y = 1|x_1, x_2) = 0.5$) and the sub-space on the right side has instances with higher probability to be positive. A rectangle indicates the region used to sample points for the corresponding bag (and a dot indicates its centroid). The top-left corner of each rectangle shows the bag index, followed by the number of instances in the bag. Bags in gray belong to class "negative" and bags in black to class "positive". In this plot we mask the class labels of the instances with the class of the corresponding bag because only the bags' class labels are observable. Note that bags can be on the "wrong" side of the instance-level decision boundary because each bag was labeled by flipping a coin based on the average class probability of the instances in it.

Note that the bag-level decision boundary is not explicitly defined but the instance-level one is. However, if the number of instances in each bag goes to infinity (i.e. basically working with the population version), then the bag-level decision boundary is defined. Since the instance-level decision boundary is symmetric w.r.t. every rectangle and so is $Pr(X|B)$, the bag-level decision boundary is defined in terms of the centroid of each bag and is the same as the instance-level one. Thus the best choice to classify a bag when given its entire population is simply to predict based on which side of the line $3x_1 + 3x_2 = 0$ the bag's centroid is on. There is also another interesting property in this asymptotic situation. Since we define $C(Y|X)$ as $Pr(Y|X)$, we can plug this into the right-hand side of Equation 3.1 to calculate the conditional expectation of $Pr(Y|X)$ given a bag b,

$$E_{X|b}[Pr(Y|X)] = \int_X Pr(Y|x)Pr(x|b)\,\mathrm{d}x$$

Assuming conditional independence between Y and b given a x,

$$\begin{aligned} &= \int_X Pr(Y|x,b)Pr(x|b)\,\mathrm{d}x \\ &= \int_X Pr(x,Y|b)\,\mathrm{d}x \\ &= Pr(Y|b) \end{aligned}$$

Thus we marginalize X in the joint distribution $Pr(X,Y)$ conditional on the existence of b and get the (conditional) prior probability: the class probability of b, $Pr(Y|b)$.

Chapter 4 presents an exact instance-based learning algorithm for this problem. It is quite obvious that the exact solution is instance-based because only the instance-level probabilities are defined. However, this artificial problem can also be tackled with other methods. We may regard $Pr(X|B)$ for each bag as a uniform distribution whose parameters are dominated by some hyper-distribution, for example, two Gaussian distributions with different mean but the same variance, one for each class. Since, as mentioned before, asymptotically the bag-level decision boundary is linear, one can imagine that this type of model is not bad for this generative model in terms of classification performance. In particular, if we model one Gaussian centered in the right-top corner in Figure 3.1 and another Gaussian in the left-bottom corner, it would be quite a good approximation of the true generative model. This thinking underlies the approach presented in Chapter 5.

	Musk 1	**Musk 2**
Bagging with Discretized PART	90.22±2.11	87.16±1.42
RBF Support Vector Machine	89.13±1.15	87.16±2.14
Bagging with Discretized C4.5	90.98±2.51	85.00±2.74
AdaBoost.M1 with Discretized C4.5	89.24±1.66	85.49±2.73
AdaBoost.M1 with Discretized PART	89.78±2.30	83.70±1.81
Discretized PART	84.78±2.51	87.06±2.16
Discretized C4.5	85.43±2.95	85.69±1.86

Table 3.1: The best accuracies (and standard deviations) achieved by the wrapper method on the Musk datasets (10 runs of stratified 10-fold cross-validation).

In the remainder of this chapter, we analyze a heuristic method based on the above generative model. Although it does not find an exact solution, it performs very well on the classification task based on this generative model. The method is very simple but the empirical performance on the Musk benchmark datasets is surprisingly good, which may be due to the similarity between these datasets and our generative model.

3.3 The Wrapper Heuristic

In this section, we briefly summarize results for a simple wrapper that, in conjunction with appropriate single-instance learning algorithms, achieves high accuracy on the Musk benchmark datasets [Frank and Xu, 2003]. Consistent with the collective assumption, this method assigns every instance the class label of the bag that it pertains to, so that a single-instance learner can learn from it. Moreover, it has two special properties: (1) at training time, instances are assigned a weight inversely proportional to the size of the bag that they belong to so that each bag receives equal weights, and (2) at prediction time, the class probability for a bag is estimated by averaging the class probabilities assigned to the individual instances in the bag. This method does not require any modification of the underlying single-instance learner as long as it generates class probability estimates and can deal with instance weights.

Table 3.1 shows the results of the wrapper method on the musk activity prediction problem for some popular single-instance learners implemented in the WEKA workbench [Witten and Frank, 1999].[2] All estimates were obtained using stratified 10-fold cross-validation (CV) repeated 10 times. The cross-validation runs were performed at the bag level.

Bagging [Breiman, 1996] and AdaBoost.M1 [Freund and Schapire, 1996] are standard single-instance ensemble methods. PART [Frank and Witten, 1998] and C4.5 [Quinlan, 1993] learn decision lists and decision trees respectively. The "RBF Support Vector Machine" is a support vector machine with a Gaussian kernel using the sequential minimal optimization algorithm [Platt, 1998]. The "discretized" versions are the same algorithms run on discretized training data using equal-width discretization [Frank and Witten, 1999]. For more details on these algorithms and how they can deal with weights and generate probability estimates, please check [Frank and Xu, 2003]. It is suffice to say that all these results are competitive with the best results achieved by other MI methods, as shown in Chapter 6.

3.4 Interpretation

Recall the wrapper method has two key features: (1) the way it assigns instance weights and class labels at training time, and (2) the probability averaging of a bag at prediction time. We provide some explanation for why this makes sense in the following.

The wrapper method is an instance-based method and like other methods in this category, it also tries to recover the instance-level probability. It is well known that many popular propositional learners aim to the minimize an expected loss function over X and Y, that is, $E_X E_{Y|X}(Loss(\beta, Y))$ where β is the parameter to be estimated and usually involved in the probability function $Pr(Y|X)$. As matter of fact, all the single-instance learning schemes in Table 3.1 are within this category.[3] Now,

[2] Some of the properties of the Musk datasets are summarized in Table 4.2 in Chapter 4.

[3] PART and C4.5 use an entropy-based criterion to select the optimal split point in a certain region.

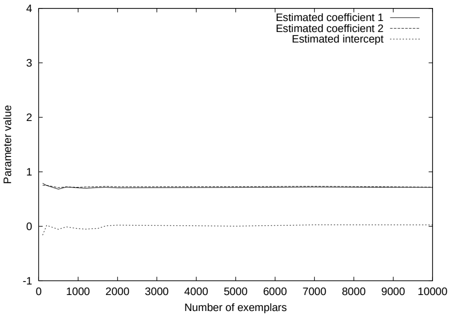


Figure 3.2: Parameter estimation of the wrapper method.

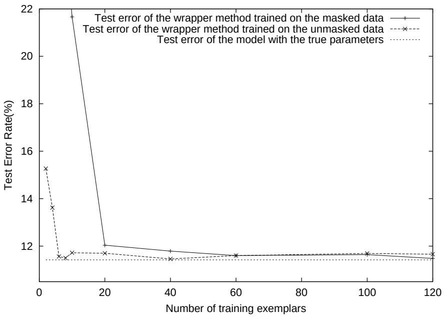


Figure 3.3: Test errors on the artificial data of the wrapper method trained on masked and unmasked data.

given a concrete bag b of size n, as defined by Equation 3.2, the sample version of $Pr(x|b)$ is $1/n$ for each instance in the bag and zero otherwise. Therefore the conditional expectation of the loss function given the presence of a bag b is simply $E_{X|b}E_{Y|X,b}(Loss(\beta, Y)) = E_{X|b}E_{Y|X}(Loss(\beta, Y))$ assuming conditional independence of Y on b given X. Plug in given $Pr(x|b)$, the conditional expected loss is simply $\sum_j \frac{1}{n} E_{y_j|x_j}(Loss(\beta, y_j))$ where x_j and y_j is the attribute vector and class label of the j^{th} instances in b respectively. We want to minimize this expected loss over all the bags, thus the final expected loss to be minimized is

$$E_B\left[E_{X|B}[E_{Y|X,B}(Loss(\beta, Y))]\right] = \sum_i \frac{1}{N} \sum_j \frac{1}{n_i} E_{y_{ij}|x_{ij}}(Loss(\beta, y_{ij})) \qquad (3.3)$$

where N is the number of bags and n_i the number of instances in bag i. Thus the weight $1/n$ of each instance x serves as $Pr(x|b)$ and the bags' weight $1/N$ is a constant outside the sum over all the bags and does not affect the minimization process of the expected loss. Nonetheless, Equation 3.3 can never be realized because y_{ij}, i.e. the class label of each instance, is *not* observable. If it were observable, this formulation could be used to find the true β. However, under the collective assumption, if we assign the bag's class label y_i to each of its instances, it may not be a bad approximation as y_i is related to all the y_{ij}. This is exactly what the wrapper method does at training time. The approximation makes the wrapper method a heuristic because it necessarily introduces bias into the probability estimates.

This is to minimize the cross-entropy or deviance loss in a piece-wise fashion.

This can be illustrated by running (weighted) linear logistic regression on the artificial data described in Section 3.2. As shown in Figure 3.2, asymptotically the estimates differ from the true parameters with a multiplicative constant.[4] It seems that the objective to recover the instance-level probability function exactly cannot be achieved with this method. Nevertheless what we really want is classification performance and it is well-known that unbiased class probability estimates are not necessary to obtain accurate classifications. In this case, since the bias is a multiplicative constant for all the parameters, the correct *decision boundary* on the instance level can be recovered. As discussed in Section 3.2, asymptotically the bag-level decision boundary is the same as the instance-level one. Thus given enough instances (normally 10 to 20) within a bag, the classification can be very accurate because the classification decision is always correctly made. This is shown in Figure 3.3. We generate a hold-out dataset of 10000 independent test bags using the same generative model described in Section 3.2. Then we use the wrapper method trained on both the “masked” data, i.e. y_{ij} is not given and y_i is assigned to every instance instead, and the “unmasked” data, i.e. y_{ij} is given for every instance. In the latter case linear logistic regression converges to the true function (convergence not shown in this thesis). At prediction time, we use the probability average of each bag, which is reasonable in this case because it is assumed in the generative model. Figure 3.3 shows that assigning a bag’s class labels to its instances does not harm classification performance. In fact, the wrapper method trained on the masked data predicts equally well as the one trained on the unmasked data using 60 training bags. It achieves the best possible accuracy (shown as the bottom line) when trained on more than 120 training bags.

The above observations are obtained in a specific artificial setting, which is much simpler than real-world problems. In general, the bias may not be a constant for all the parameters so even the true decision boundary cannot be recovered. However, practical generative models may have factors that restrict, to some extent, the bias or the harmful effect of the bias on the classification. For example, we observed

[4] The reason the intercept estimate seems unbiased is that the true value is 0 and with a multiplicative bias, the estimate is still 0.

from the artificial data that the less area each bag occupies in the instance space, the less bias the wrapper method has. When the ranges of bags become small, the bias is literally negligible. Hence if there are some restrictions on the range of a bag in the instance space, the wrapper method can work well. Indeed, we observed that on the Musk datasets, the in-bag variances are very small for most of the bags, which may explain the feasibility of the wrapper method. Intuitively, this heuristic will work well if the true class probabilities $Pr(Y|X)$ are "similar" for all the instances in a bag (under the above generative model) because we use the bags' class labels to approximate those of the corresponding instances. Therefore in general, as long as this condition is approximately correct, no matter by what means, the wrapper method can work.

Finally what the wrapper method does at prediction time is reasonable assuming the above generative model. Nevertheless, it seems that the wrapper method does not use the assumption of a bag's class probability being the averaged instances' class probabilities at training time. In fact, by assigning the bags' class labels to their instances, it only assumes the general collective assumption but not any specific assumption regarding how the bags' class labels are created. Therefore we can use other methods at prediction time such as taking the normalized geometric average of the instances' class probabilities within a bag as the bag's probability, as long as the method is consistent with the general collective assumption. However, according to our experience, taking the arithmetic average of the instances' probability for a bag is more robust on practical datasets like the Musk datasets. Thus we recommend this method for the wrapper method in general.

As explained, the wrapper method is only a heuristic method that can work well in practice under some conditions. At least for the specific generative model proposed in Section 3.2 of this chapter, it produces accurate classifications, although it is not good for probability estimation. In Chapter 4, we will present an exact algorithm that can give accurate probability estimates for the same generative model. They are also based on the normal single-instance learning schemes and aim to upgrade them to deal with MI data. The disadvantage of such an approach is that it heavily relies on exact assumptions at the training time. Thus the significant modifications of the underlying propositional learning algorithms are inevitable. The wrapper method, on the other hand, is much simpler and more convenient.

3.5 Conclusions

In this chapter we first introduced a new assumption other than the MI assumption for MI learning — the collective assumption. This assumption regards the class label of a bag related to all the instances within a bag. We then showed some concrete applications of the collective assumption to exactly generate MI data. One of the applications is to take the averaged probability of all the instances in the same bag as the class probability of that bag.

Under the above exact generative model, we further assumed that the instances within the same bag have similar class probabilities. Consequently the probability of a bag is also similar to those of its instances. These assumptions allow us to develop a heuristic wrapper method for MI problems. This method wraps around normal single-instance learning algorithms by (1) assigning the class label of a bag and instance weights to the instances at training time, and (2) averaging instances' class probabilities of a bag at testing time.

Assigning bags' class labels to their corresponding instances and averaging the instances' class probabilities are the application of the above two assumptions (the collective assumption and the "similar probability" assumption). The instances' weights and the wrapping scheme were motivated based on the instance-level loss function (encoded in the single-instance learners) over all the bags. We also showed an artificial example where this wrapper method can perform well for classification although its probability estimates are biased. Empirically, we found out that this method works very well with the Musk benchmark datasets, in spite of its simplicity.

Chapter 4

Upgrading Single-instance Learners

Among many solutions to tackle multiple instance (MI) learning problems, one approach has become increasingly popular, that is, to upgrade paradigms from the normal single-instance learning to deal with MI data. The efforts described in this chapter also fall into this category. However, unlike most of the current algorithms within this category, we adopt an assumption-based approach that is based on the statistical decision theory. Starting with analyzing the assumptions and the underlying generative models of MI problems, we provide a fairly general and justified framework for upgrading single-instance learners to deal with MI data. The key feature of this framework is the minimization of the expected bag-level loss function based on some assumptions. As an example we upgrade two popular single-instance learners, linear logistic regression and AdaBoost and test their empirical performance. The assumptions and underlying generative models of these methods are explicitly stated.

4.1 Introduction

The motivating application for MI learning was the drug activity problem considered by Dietterich *et al.* [1997]. The generative model for this problem was basically regarded as a two-step process. Dietterich *et al.* assumed there is an Axis-Parallel

Rectangle (APR) in the instance space that accounts for the class label of each instance (or each point in the instance space). Each instance within the APR is positive and all others negative. In the second step, a bag of instances is formed by sampling (not necessarily randomly) from the instance space. The bag's class label is determined by the MI assumption, i.e., a bag will be positive if at least one of its instances is positive (within the assumed APR) and otherwise negative. With this perspective, Dietterich *et al.* proposed APR algorithms that attempts to find the best APR under the MI assumption. They showed empirical results of the APR methods on the Musk datasets, which represent a musk activity prediction problem.

In this chapter, we follow the same perspective as that in [Dietterich et al., 1997] but with different assumptions. We adopt an approach based on the statistical decision theory and select assumptions that are well-suited for upgrading each single-instance learner.

The rest of the chapter is organized as follows. In Section 4.2 we explain the underlying generative model we assume and show artificial MI data generated using this model. In Section 4.3 we describe a general framework for upgrading normal single-instance learners according to the generative model. We also provide two examples of how to upgrade the linear logistic regression and AdaBoost [Freund and Schapire, 1996] within this framework. These methods have not been studied in the MI domain before. Section 4.4 shows some properties of our methods on both the artificial data and practical data. Some regularization techniques, as used in normal single-instance learning, will also be introduced in an MI context. In Section 4.5 we show that the methods presented in this chapter perform comparatively well on the benchmark datasets, i.e. the Musk datasets. Section 4.6 summarizes related work and Section 4.7 concludes this chapter.

4.2 The Underlying Generative Model

The basis for the work presented in this chapter was an analysis of the generative model (implicitly) assumed by Dietterich *et al.*. The result of this analysis was that the generative model is actually modeled as a two-step process. The first step is an instance-level classification process and the second step determines the class labels of a bag based on the first step and the MI assumption. As a matter of fact, the only probabilistic algorithm in MI learning, the Diverse Density (DD) [Maron, 1998] algorithm, followed the same line of thinking. In the first step, DD assumes a radial (or "Gaussian-like") formulation for the true posterior probability function $Pr(Y|X)$. In the second step, based on the values of $Pr(Y|X)$ of all the instances within a bag, it assumes either a multi-stage (as in the noisy-or model) or a one-stage (as in the most-likely-cause model) Bernoulli process to determine the class label of a bag. Therefore DD amounts to finding one (or more) Axis-Parallel hyper-Ellipse (APE)[1] under the MI assumption.

It is natural to extend the above process of generating MI data to a more general framework. Specifically, as in single-instance learning, we assume a joint distribution over the feature variable X and the class (response) variable Y, $Pr(X, Y) = Pr(Y|X)Pr(X)$. The posterior probability function $Pr(Y|X)$ determines the instance-level decision boundary that we are looking for. However, in MI learning we introduce another variable B, denoting the bags, and what we really want is $Pr(Y|B)$. Let us assume that given a bag b and all its instances x_b, $Pr(Y|b)$ is a function of $Pr(Y|x_b)$, i.e., $Pr(Y|b) = g(Pr(Y|x_b))$. The form of $g(.)$ is determined based on some assumptions. In the APR algorithms [Dietterich et al., 1997], the instance-level decision boundary pattern is modeled as an APR and $g(.)$ is an instance-selection function based on the MI assumption. In DD [Maron, 1998], the instance-level decision boundary pattern is an APE and $g(.)$ is either the noisy-or or the most-likely-cause model, which are both due to the MI assumption. Therefore the "APR-like pattern and MI assumption" combination is simply a special case of

[1] Note that an APE is very similar to an APR but that it is differentiable.

our framework.

In this chapter we are creating new combinations within this framework but with different assumptions. We change the decision boundary to other patterns and substitute the MI assumption with the collective assumption introduced in Chapter 3. We model the instance-level class label based on $Pr(Y|X)$ or its logit transformation, i.e. the log-odds function $\log \frac{Pr(Y=1|X)}{Pr(Y=0|X)}$, because many normal single-instance learners aim to estimate them and we are aiming to upgrade these algorithms. We can think of a bag b as a certain area in the instance space. Then given b with n instances, we have

$$Pr(Y|b) = \frac{1}{n}\sum_{i}^{n} Pr(y|x_i) \tag{4.1}$$

or

$$\log \frac{Pr(Y=1|b)}{Pr(Y=0|b)} = \frac{1}{n}\sum_{i=1}^{n} \log \frac{Pr(Y=1|x_i)}{Pr(Y=0|x_i)}$$
$$\Rightarrow \begin{cases} Pr(Y=1|b) = \frac{[\prod_{i=1}^{n} Pr(y=1|x_i)]^{1/n}}{[\prod_{i=1}^{n} Pr(y=1|x_i)]^{1/n}+[\prod_{i=1}^{n} Pr(y=0|x_i)]^{1/n}} \\ Pr(Y=0|b) = \frac{[\prod_{i=1}^{n} Pr(y=0|x_i)]^{1/n}}{[\prod_{i=1}^{n} Pr(y=1|x_i)]^{1/n}+[\prod_{i=1}^{n} Pr(y=0|x_i)]^{1/n}} \end{cases} \tag{4.2}$$

where $x_i \in b$. Even though the assumptions are quite intuitive, we provide a formal derivation in the following.

As discussed in Section 3.1 of Chapter 3, we define two versions of generative models — the population version and the sample version. In the population version, the *conditional density* of the feature variable X given a bag $B = b$ is

$$Pr(x|b)^2 = \begin{cases} \frac{Pr(x)}{\int_{x \in b} Pr(x)\, \mathrm{d}x} & \text{if } x \in b, \\ 0 & \text{otherwise}. \end{cases} \tag{4.3}$$

[2] Note that we abuse the term $Pr(.)$ here because for a numeric feature, what we have is a density function of X instead of the probability.

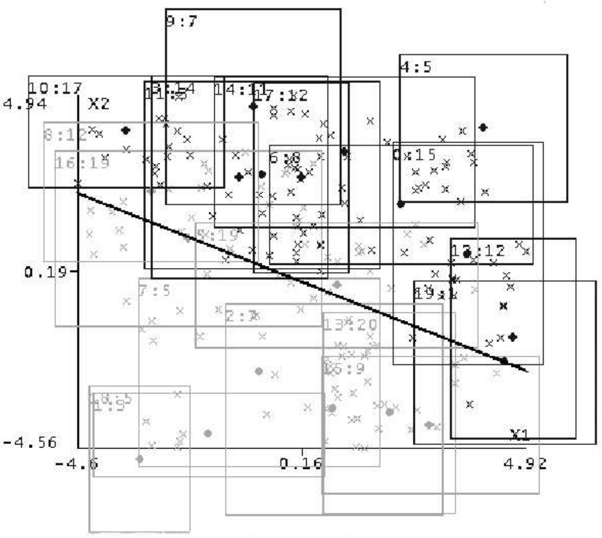


Figure 4.1: An artificial dataset with 20 bags.

And in the sample version,

$$Pr(x|b) = \begin{cases} \frac{1}{n} & \text{if } x \in b, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

where n is the number of instances inside b. This is true no matter what distribution $Pr(X)$ is, as long as the instances are randomly sampled. Then, for any formulation of the class label property of a bag $C(Y|b)$, according to our collective assumption we associate it with the instances by calculating the *conditional expectation* over X, which results in Equation 3.1 discussed in the second question in Section 3.1 of Chapter 3.

In the sample version, we substitute $Pr(x|b)$ in Equation 3.1 with that in Equation 4.4 and use the sum instead of the integral. If $C(Y|.)$ is $Pr(Y|.)$, then we get Equation 4.1, which is the arithmetic average of the corresponding instance probabilities. If $C(Y|.)$ is $\log \frac{Pr(Y=1|.)}{Pr(Y=0|.)}$, then we obtain Equation 4.2, which is the normalized geometric average of the instance probabilities. Note that in this model introducing the bags B does *not* change the joint distribution $Pr(X, Y)$. It only casts a new condition so that the class labels of the instances are “masked” by the “collective” class label.

The above generative models are better illustrated by an artificial dataset, which will also be used in later sections. We consider an artificial domain with two independent attributes. More specifically, we used the same mechanism for generating artificial data as that in Section 3.2 of Chapter 3 except that we changed the density of X, $Pr(X)$ and use a different linear logistic model. Now the density function, along each dimension, is a triangle distribution instead of a uniform distribution:

$$f(x) = 0.2 - 0.04|x|$$

And the instance-level class probability was defined by the linear logistic model of

$$Pr(y = 1|x_1, x_2) = \frac{1}{1 + e^{-x_1 - 2x_2}}$$

Thus the instance-level decision boundary pattern is still a hyperplane, but different from the one modeled in the artificial data in Section 3.2. We changed these in order to demonstrate that our framework can deal with any form of $Pr(X)$ and $Pr(Y|X)$, as long as the correct family of $Pr(Y|X)$ (in this case the linear logistic family) and the correct underlying assumption (in this case the collective assumption) are chosen. Finally we took Equation 4.2 to calculate $Pr(y|b)$. Again we labeled each bag according to its class probability. The class labels of the instances are not observable.

Now we have constructed a dataset based on the “Hyperplane (or linear boundary) and the collective Assumption” combination instead of the “APR-like (or quadratic boundary) and MI Assumption” combination used in the APR algorithms [Dietterich et al., 1997] and DD [Maron, 1998].

Figure 4.1 shows a dataset with 20 bags that was generated according to this generative model. Same as in Chapter 3, the black line in the middle is the instance-level decision boundary (i.e. where $Pr(y = 1|x_1, x_2) = 0.5$) and the sub-space on the right side has instances with higher probability to be positive. A rectangle indicates the region used to sample points for the corresponding bag (and a dot indicates its centroid). The top-left corner of each rectangle shows the bag index, followed by the number of instances in the bag. Bags in gray belong to class “negative” and bags in black to class “positive”. Note that bags can be on the “wrong” side of the instance-level decision boundary because each bag was labeled by flipping a coin based on the normalized geometric average class probability of the instances in it.

4.3 An Assumption-based Upgrade

In this section we first show how to solve the above problem in the artificial domain. Since the generative model is a linear logistic model, we can upgrade linear logistic regression to solve this problem exactly. Then we generalize the underlying ideas to a general framework to upgrade single-instance learners based on some assumptions, which also includes the APR algorithms [Dietterich et al., 1997] and the DD algorithm [Maron, 1998]. Finally within this framework we also show how to upgrade AdaBoost algorithm [Freund and Schapire, 1996] to deal with MI data.

First we upgrade linear logistic regression together with the collective assumption so that it can deal with MI data. Note that normal linear logistic regression can no longer apply here because the class labels of instances are masked by the “collective” class label of a bag. Suppose we knew what exactly the collective assumption is, say, Equation 4.2, then we could first construct the probability $Pr(Y|b)$ using Equation 4.2, and estimate the parameters (the coefficients of attributes in this case) using the standard maximum binomial likelihood method. In this way we fully “recover” the instance-level probability function in spite of the fact that the class labels are masked. When a test bag is seen, we can calculate the class probability $Pr(Y|b_{test})$ according to the “recovered” probability estimate and the same assumption we used at training time. The classification is based on $Pr(Y|b_{test})$. Mathematically, in the logistic model, $Pr(Y=1|x) = \frac{1}{1+\exp(-\beta\mathbf{x})}$ and $Pr(Y=0|x) = \frac{1}{1+\exp(\beta\mathbf{x})}$ where β is the parameters to be estimated. According to

Equation 4.2, we construct:

$$\begin{cases} Pr(Y=1|b) = \frac{[\prod_i^n Pr(y=1|x_i)]^{1/n}}{[\prod_i^n Pr(y=1|x_i)]^{1/n}+[\prod_i^n Pr(y=0|x_i)]^{1/n}} = \frac{\exp(\frac{1}{n}\beta\sum_i \mathbf{x_i})}{1+\exp(\frac{1}{n}\beta\sum_i \mathbf{x_i})} \\ Pr(Y=0|b) = \frac{[\prod_i^n Pr(y=0|x_i)]^{1/n}}{[\prod_i^n Pr(y=1|x_i)]^{1/n}+[\prod_i^n Pr(y=0|x_i)]^{1/n}} = \frac{1}{1+\exp(\frac{1}{n}\beta\sum_i \mathbf{x_i})} \end{cases}$$

Then we model the class label determination process of each bag as a one stage Bernoulli process. Thus the binomial log-likelihood function is:

$$LL = \sum_{i=1}^{N} [Y_i \log Pr(Y=1|b) + (1-Y_i)\log Pr(Y=0|b)] \qquad (4.5)$$

where N is the number of bags. By maximizing the likelihood function in Equation 4.5 can we estimate the parameters β. Maximum likelihood estimates (MLE) are known to be asymptotically unbiased, as illustrated in Section 4.4. This formulation is based on the assumption of Equation 4.2. In practice, it is impossible to know the underlying assumptions so other assumptions may also apply, for instance, the assumption of Equation 4.1. In that case, the log-likelihood function of Equation 4.5 remains unchanged but the formulation of $Pr(Y|b)$ is changed to Equation 4.1. We call the former method “MILogisticRegressionGEOM” and the latter “MILogisticRegressionARITH” in this chapter.

As usual, the maximization of the log-likelihood function is carried out via numeric optimization because there is no analytical form of the solution in our model.[3] Based on the “Rule of Parsimony”, we want as few parameters as possible. In the case of linear logistic regression, we only search for parameter values around zero. Thus the linear pattern means that only local optimization is needed, which saves us great computational costs. The radial formulation in DD [Maron, 1998], on the other hand, implies a complicated global optimization problem.

In general, since many single-instance learners amount to minimizing the expected loss function $E_X E_{Y|X}[Loss(X,\beta)]$ in order to estimate the parameter β in $Pr(Y|X)$, we can upgrade any normal single-instance learner that factorizes $Pr(X,Y)$ into $Pr(Y|X)Pr(X)$ and estimates $Pr(Y|X)$ directly. This category covers a wide

[3] The choice of numeric optimization methods used in this thesis is discussed in Chapter 7.

range of single-instance learners, thus the method is quite general. It involves four steps:

1. Based on whatever assumptions believed appropriate, build a relationship between the class probability of a bag b, $Pr(Y|b)$, and that of its instances x_b, $Pr(Y|x_b)$, i.e. $Pr(Y|b) = g(Pr(Y|x_b))$. Since $Pr(Y|X)$ is usually defined by the single-instance learner under consideration, the only thing to decide is $g(.)$.

2. Construct a loss function at the bag level and take the expectation over all the bags instead of over instances, i.e. the expected loss function is now $E_B E_{Y|B}[Loss(B, \beta)]$. Note that the parameter vector β is the instance-level parameter vector because it determines $Pr(Y|X)$ (or its transformations).

3. Minimize the bag-level loss function to estimate β.

4. When given a new bag b_{test}, first calculate $Pr(Y|x_{test}, \hat{\beta})$ and then calculate $Pr(Y|b_{test}) = g(Pr(Y|x_{test}, \hat{\beta}))$ based on the same assumption used in Step 1. Then classify b_{test} according to whether $Pr(Y|b_{test})$ is above 0.5.

The negative binomial log-likelihood is a loss function (also called deviance or cross-entropy loss). Thus the above two MI linear logistic regression methods fit into this framework. They use the linear logistic formulation for $Pr(Y|X)$ and the collective assumption. The Diverse Density (DD) algorithm [Maron, 1998] also uses the maximum binomial likelihood method thus it can be easily recognized as a member of this framework. It uses a radial formulation for $Pr(Y|X)$ and the MI assumption. The APR algorithms [Dietterich et al., 1997], on the other hand, directly minimize the misclassification error loss at the bag level and the bounds of the APR are the parameters to be estimated for $Pr(Y|X)$. Since the APR algorithms regard the instance-level classification as a deterministic process, $Pr(Y = 1|x)$ can be regarded as 1 for any instance x within the APR. Otherwise $Pr(Y = 1|x) = 0$. Note that in this framework, significant changes to the underlying single-instance algorithms seems inevitable. This is in contrast to the heuristic wrapper method presented in Chapter 3.

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

Table 4.1: The upgraded MI AdaBoost algorithm.

Linear logistic regression and the quadratic formulation (as in DD) assume a limited family of underlying patterns. There are more flexible single-instance learners like boosting and the support vector machine (SVM) algorithms that can model larger families of patterns. However, the general upgrading framework presented above means that, under certain assumptions, one can model a wide range of decision boundary patterns. Here we provide an example of how to upgrade the AdaBoost algorithm into an MI learner based on the collective assumption. Intuitively AdaBoost can easily be wrapped around an MI algorithm (without changes to the AdaBoost algorithm), but since there are not many "weak" MI learners available we are more interested in taking the single-instance learners as the base classifier of the upgraded method.

AdaBoost originated in the Computational Learning Theory domain [Freund and Schapire, 1996], but received a statistical explanation later on [Friedman, Hastie and Tibshirani, 2000]. It can be shown that it aims to minimize an exponential loss function in a forward stagewise manner and ultimately estimates $\frac{1}{2} \log \frac{Pr(Y=1|X)}{Pr(Y=-1|X)}$ (based on an additive model) [Friedman et al., 2000].[4] Now, under the collective assumption, we use the Equation 4.2 because the underlying single-instance learner

[4] Note that in AdaBoost, class labels are coded into $\{-1, 1\}$ instead of $\{0, 1\}$.

(i.e. normal AdaBoost) estimates the log-odds function (We could also use the standard MI assumption here, but this would necessarily introduce the max function, which makes the optimization much harder). We first describe the upgraded AdaBoost algorithm in Table 4.1 and then briefly explain the derivation. The notation used in Table 4.1 is as follows: N is the number of bags and we use the subscript i to denote the i^{th} bag, where $i = 1, 2, \ldots, N$. Suppose there are n_i instances within the i^{th} bag. Then we use the subscript j to refer to the j^{th} instance, where $j = 1, 2, \ldots, n_i$. Therefore x_{ij} denotes the j^{th} instance in the i^{th} bag.

The derivation follows exactly the same line of thinking as that in [Friedman et al., 2000]. In the derivation below we regard the expectation sign E as the sample average instead of the population expectation. We are now looking for a function over all the bags $F(B)$ that minimizes $E_B E_{Y|B}[\exp(-yF(B))]$ where given a bag b,

$$F(b) = \sum_n F(x_b)/n. \tag{4.6}$$

We want to expand $F(B)$ to $F(B) + cf(B)$ in each iteration with the restriction $c > 0$. First, given $c > 0$ and the current bag-level weights $W_B = \exp(-yF(B))$, we are searching for the best $f(B)$. After second order expansion of $\exp(-ycf(B))$ about $f(B) = 0$, we are seeking the maximum of $E_W[yf(B)]$. If we had an MI base learner in hand, we could estimate $f(B)$ directly. However we are interested in wrapping around a single-instance learner, thus we expand $f(B)$ for each bag b according to Equation 4.6: $f(b) = \sum_n h(x_b)/n$ where $h(x_b) \in \{-1, 1\}$. Now we are seeking $h(.)$ to maximize

$$E_W[yh(x_b)/n] = \sum_{i=1}^{N} \sum_{j}^{n_i} [\frac{1}{n_i} W_i Pr(y = 1|b_i) h(x_{ij}) - \frac{1}{n_i} W_i Pr(y = -1|b_i) h(x_{ij})]$$

The solution is

$$h(x_{ij}) = \begin{cases} 1 & \text{if } \frac{W_i}{n_i} Pr(y = 1|b_i) - \frac{W_i}{n_i} Pr(y = -1|b_i) > 0 \\ -1 & \text{otherwise} \end{cases}$$

This formula simply means that we are looking for the function $h(.)$ at the instance level such that given a weight of $\frac{W_i}{n_i}$ for each instance, its value is determined by the probability $Pr(y|b_i)$. Note that this probability is the same for all the instances in the bag. Since the class label of each bag reflects its probability, we can assign the class label of a bag to its instances. Then, with the weights $\frac{W_i}{n_i}$, we use a single-instance learner to provide the value of $h(.)$. This constitutes Step 2a in the algorithm in Table 4.1. Note that in Chapter 3, we proposed a wrapper method to apply normal single-instance learners to MI problems. There, it is a heuristic to assign the class label of each bag to the instances pertaining to it because we minimize the loss function within the underlying (instance-level) learner. Since the underlying learners are at the instance level, they require the class label for each instance instead of each bag. That is why it is a heuristic. Here we only use the underlying instance-level learners to estimate $h(.)$, and our *objective* is to estimate $h(.)$ such that $yh(x_b)/n$ is maximized over all the (weighted) instances, where y is the *bag's* class label. Therefore assigning the class label of a bag to its instances is actually our *aim* here because we are trying to minimize a bag-level loss function. Hence it is *not* a heuristic or approximation in this method.

Next, if we average $-y_i h(x_{ij})$ for each bag, we get $-y_i f(b_i) = 2e_i - 1$, where $e_i = \sum_j 1_{(h_m(x_{ij}) \neq y_i)}/n_i$ and this is Step 2b in the algorithm. Then given $-yf(B) \in [-1, 1]$ (more precisely, $-yf(B) \in \{-1, -1 + 1/n, -1 + 2/n, \cdots, 1 - 2/n, 1 - 1/n, 1\}$) we are looking for the best $c > 0$. To do this we can directly optimize the objective function

$$\begin{aligned} E_B E_{Y|B}[\exp(F(B) + c \times (-yf(B)))] &= \sum_i W_i \exp[c_m \times (-\frac{y \sum_j h(x_{ij})}{n_i})] \\ &= \sum_i W_i \exp[(2e_i - 1)c_m]. \end{aligned}$$

This constitutes Step 2d.

This objective function will not have a global minimum if all $e_i < 0.5$. The interpretation is analogous to that in the mono-instance AdaBoost. In normal AdaBoost, the objective function has no global minimum if every instance is correctly classified based on the sign of $f(x)$ (i.e. zero misclassification error is achieved). Here we can classify a *bag* b according to $f(b) = \sum_n h(x_b)/n$. Then $-y_i f(b_i) = 2e_i - 1$ simply means that if the error rates within all the bags e_i are less than 0.5, all the bags will be correctly classified because all the $f(b_i)$ will have the same sign as y_i. Thus we have reached zero error and no more boosting can be done — as in normal AdaBoost. Therefore we check this in Step 2c.

The solution of this optimization problem may not have an easy analytical form. However, it is a one-dimensional optimization problem. Hence the Newton family of optimization techniques can find a solution in super-linear time [Gill, Murray and Wright, 1981] and the computational cost is negligible compared to the time to build the "weak" classifier. Therefore we simply search for c using a Quasi-Newton method in Step 2d.

Note that c is not necessarily positive. If it is negative, we can simply reverse the prediction of the weak classifier and get a positive c (this can be done automatically in Step 2f). However, we bow to the AdaBoost convention and restrict c to be positive. Thus we check that in Step 2e.

Finally we update the bag-level weights in Step 2f according to the additive structure of $F(B)$, in the same way as in normal AdaBoost. Note that if a bag has more misclassified instances in it, it gets a higher weight in the next iteration, which is intuitively appealing. Another appealing property of this algorithm is that if there is only one instance per bag, i.e. the data is actually single-instance data, this algorithm naturally degrades to normal AdaBoost. To see why, note that the solution for c in Step 2d will be exactly $\frac{1}{2}\log\frac{1-err_w}{err_w}$ where err_w is the weighted error. Hence the weight update will also be the same as in AdaBoost.

It is easy to see that to classify a test bag, we can simply regard $F(B)$ as the bag-level log-odds function and take Equation 4.2 to make a prediction.

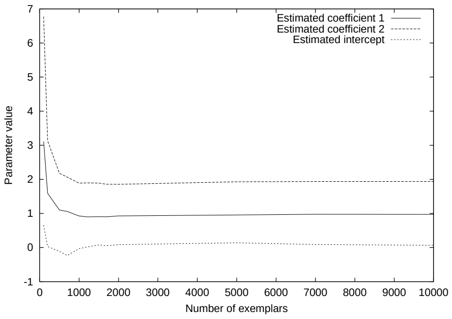


Figure 4.2: Parameter estimation of the MILogisticRegressionGEOM on the artificial data.

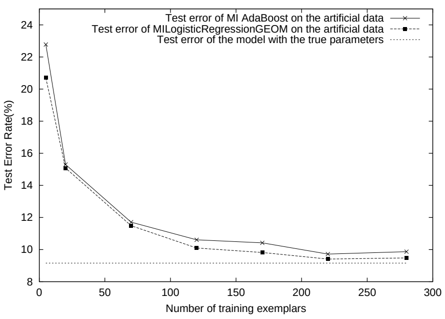


Figure 4.3: Test error of MILogisticRegressionGEOM and the MI AdaBoost algorithm on the artificial data.

4.4 Property Analysis and Regularization Techniques

In this section we first show some properties of the upgraded MI learners. We use the artificial data generated according to the scheme in Section 4.2 to show some asymptotic properties of the probability estimates using an MI logistic regression algorithm. Next we show the test error of the MI AdaBoost algorithm on the artificial data to see whether it can perform well on classifying MI data. Finally, since the "number of iterations" property of the boosting algorithms is usually of interest, we show this property for our MI AdaBoost algorithm against the training and test errors on the Musk1 dataset. We observe that its behavior is very similar to its single-instance predecessor. After analyzing the properties we introduce some regularization methods for the above algorithms in order to increase generalization power on practical datasets.

Because we know the artificial data described in Section 4.2 is generated using a linear logistic model based on the normalized geometric average formulation from Equation 4.2, "MILogisticRegressionGEOM" is the natural candidate to deal with this specific MI dataset. Figure 4.2 shows the parameters estimated by this method when the number of bags increases. It can be seen that the estimates converge to the true parameters asymptotically. The more training bags, the more stable the estimates. This is a consequence of the fact that the class probability estimates of this algorithm are consistent (or asymptotically unbiased) by virtue of the maxi-

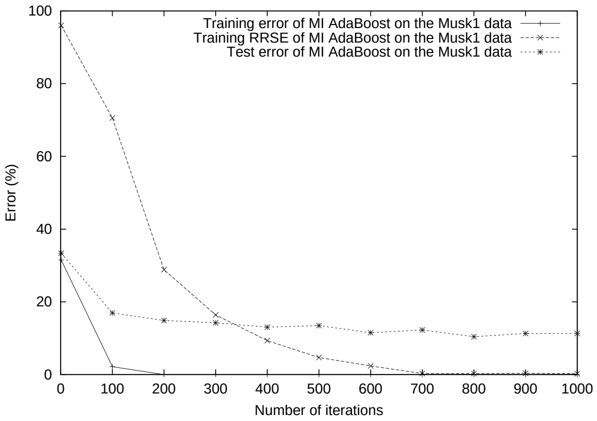


Figure 4.4: Error of the MI AdaBoost algorithm on the Musk1 data.

mum likelihood method. The consistency of the MLE can be proven under fairly general conditions [Stuart, Ord and Arnold, 1999]. In this specific artificial data, we used a triangle distribution for the density of X, as explained in Section 4.2. However, we observed that the exact form of this density does not matter. This is reasonable because our model only assumes random sampling from the distribution corresponding to a bag. The exact form of $Pr(X)$ does not matter.

MILogisticRegressionGEOM successfully "recovers" the instance-level class probability function (and the underlying assumption also holds for the test data). Consequently we expect it to achieve the optimal error rate on this data. As explained before, the MI AdaBoost algorithm is also based on Equation 4.2, thus it is also expected to perform well in this case. To test this, we first generate an independent test dataset of 10000 bags, then generate training data (with different random seeds) for different numbers of training bags. The decision stumps are used as the weak learner and the maximal number of boosting iterations was set to 30. The test error of both methods on the test data against different numbers of bags is plotted in Figure 4.3. When the number of training bags increases, the error rate goes closer to the best error rate and eventually both methods accomplish close to the "perfect" performance. However, unlike MILogisticRegressionGEOM, MI AdaBoost cannot achieve the exact best error rate, mainly because in this case we are approximating a line (the decision boundary) using axis-parallel rectangles, only based on a finite amount of data.

Finally, people are often interested in how many iterations are needed in the boosting algorithms and it is also an issue in our MI AdaBoost algorithm. Cross Validation (CV) is a common way to find the best number of iterations. It is known that even after the training error ceases to decrease, it is still worthwhile boosting in order to increase the confidence (or *margin*) of the classifier [Witten and Frank, 1999]. In the two-class case (as in the Musk datasets), increasing the margin is equivalent to reducing the estimated root relative squared errors (RRSE) of the probability estimates. It turns out that this statement also holds well for MI AdaBoost. As an example, we show the results of MI AdaBoost on the Musk1 dataset in Figure 4.4. The training error is reduced to zero after about 100 iterations but the RRSE keeps decreasing until around 800 iterations. The test error, averaged over 10 runs of 10-fold CV, also reaches a minimum at around 800 iterations, which is 10.44% (standard deviation: 2.62%). After 800 iterations, the RRSE does not seem to improve any further and the test error rises due to overfitting. The error rate is quite low for the Musk1 dataset, as will be shown in Section 4.5. However, boosting on decision stumps is too slow for the Musk2 dataset. We observed that the RRSE does not settle down even after 8000 iterations. Therefore it is computationally too expensive to be used in practice and we present results based on regularized decision trees as the “weak” classifier.

Regularization is commonly used in single-instance learning. It introduces a bias in the probability estimates in order to increase the generalization performance by reducing the chance of overfitting. It works because the underlying assumptions of a learner rarely perfectly hold in practice. In MI learning, regularization can be naturally inherited from the corresponding single-instance learners if we upgrade one of them. For example, in the MI support vector machine (SVM) [Gärtner et al., 2002], there is a model complexity parameter that controls the scale of the regularization. For non-standard methods like the APR algorithms, other ways like kernel density estimation (KDE) [Dietterich et al., 1997] are adopted to achieve the function of regularization. We can also impose regularization in our methods as follows.

For single-instance linear logistic regression, a shrinkage method is proposed in [le Cessie and van Houwelingen, 1992]. We adopt this “ridge regression” method here. Thus we simply add an L_2 penalty term $\lambda||\beta||^2$ to the likelihood function in Equation 4.5 where λ is the ridge parameter. Note that in ridged logistic regression — since it is not invariant along each dimension — we need to standardize the data to zero-mean/unit-standard-deviation before fitting the model, and transform the model back after fitting [Hastie et al., 2001]. The mean and standard deviation used in the standardization in our MI logistic regression methods are estimated using the weighted instance data, with each instance weighted by the inverse of the number of instances in the corresponding bag. This is done because intuitively we would like to have equal weight for every bag. Note that unlike some current MI methods that change the data, we do not pre-process the data — the standardization is simply part of the algorithm.

In boosting with decision trees, both the tree size and the number of iterations determine the degrees of freedom, and there are several ways to regularize it in single-instance learning [Hastie et al., 2001]. We use C4.5 [Quinlan, 1993] which does not have an explicit option to specify how many nodes will be in a tree, we use an alternative way to shrink the tree size, that is, we specify a larger minimal number of (weighted) instances for the leaf nodes (the default setting in C4.5 is two). Together with the restriction of the number of iterations we can achieve a very coarse form of regularization in our MI AdaBoost algorithm. By enlarging the minimal number of leaf-node instances and in turn shrinking the tree size, we effectively make the tree learner “weaker”. We will only show experimental results for the MI AdaBoost algorithm based on these regularized trees.

4.5 Experimental Results

This section we present experimental results of our MI algorithms on the Musk benchmark datasets [Dietterich et al., 1997]. As already discussed in Chapter 1, there are two overlapping datasets describing the musk activity prediction problem, namely the Musk1 and Musk2 data. Some of the properties of the datasets are

	Musk 1	**Musk 2**
Number of bags	92	102
Number of attributes	166	166
Number of instances	476	6598
Number of positive bags	47	39
Number of negative bags	45	63
Average bag size	5.17	64.69
Median bag size	4	12
Minimum bag size	2	1
Maximum bag size	40	1044

Table 4.2: Properties of the Musk 1 and Musk 2 datasets.

summarized in Table 4.2.

Table 4.3 shows the performance of related MI methods on the Musk datasets, as well as that of our methods developed in this chapter. The evaluation is either by leave-one-out (LOO) or by 10-fold Cross Validation (CV) *at the bag level*. In the first part we include some of the current solutions to MI learning problems. All the methods that depend on the “APR-like pattern and MI assumption” combination are shown. They are: the best of the APR algorithms, iterated-discrim APR with KDE [Dietterich et al., 1997], the Diverse Density algorithm [Maron, 1998][5] and the MULTINST algorithm [Auer, 1997]. Although MI Neural Networks [Ramon and Raedt, 2000] do not model the APR-like pattern exactly, they depend heavily on the MI assumption and followed a thinking very similar to the above methods. The pattern that they model really depends on the complexity of the networks that are built. Thus we also include it. It can be seen that the models based on the “linear pattern and the collective assumption” combination can also achieve comparably good results on the Musk datasets. Therefore we believe that in practice what really matters is the right combination of the decision boundary pattern and

[5] Note that we do not include the EM-DD algorithm [Zhang and Goldman, 2002] here even though it was reported to have the best performance on the Musk datasets. We do so for two reasons: 1. There were some errors in the evaluation process of EM-DD; 2. It can be shown via some theoretical analysis and an artificial counter-example (Appendix D) that EM-DD cannot find a maximum likelihood estimate (MLE) in general due to the (tricky) likelihood function it aims to optimize. Since the DD algorithm is a maximum likelihood method, the solution that EM-DD finds cannot be trusted if it fails to find the MLE.

Methods	**Musk 1**		**Musk 2**	
	LOO	**10CV**	**LOO**	**10CV**
iterated-discrim APR with KDE	–	7.6	–	10.8
maxDD	–	11.1	–	17.5
MULTINST	–	23.3	–	16.0
MI Neural Networks[a]	–	12.0	–	18.0
SVM with the MI kernel	13.0	13.6±1.1	7.8	12.0±1.0
Citation-kNN	7.6	–	13.7	–
MILogisticRegressionGEOM	13.04	14.13±2.23	17.65	17.74±1.17
MILogisticRegressionARITH	10.87	13.26±1.83	16.67	15.88±1.29
MI AdaBoost with 50 iterations	10.87	12.07±1.95	15.69	15.98±1.31

[a] It was not clear which evaluation method the MI Neural Network used. We put it into 10CV column just for convenience.

Table 4.3: Error rate estimates from either 10 runs of stratified 10-fold cross-validation or leave-one-out evaluation. The standard deviation of the estimates (if available) is also shown.

the assumption. Simple patterns and assumptions together may be sufficient.

There are many methods that aim to upgrade single-instance learners to deal with MI data, as described in Chapter 2. Although they are definitely not in the same category as the methods in the first part, and their assumptions and generative models are not totally clear, we include some of them in the second part. Because there are too many to list here, we simply chose the ones with the best performance on the Musk datasets: the SVM with the MI kernel [Gärtner et al., 2002] and an MI K-Nearest-Neighbour algorithm, Citation-kNN [Wang and Zucker, 2000]. For the SVM with the MI kernel, the evaluation is via leave-10(bags)-out, which is similar to that of 10-fold CV because the total number of bags in either dataset is close to 100. Thus we put its results into the "10CV" columns.

Finally in the third part we present results for the methods from this chapter. Since we introduced regularization, as also used by some other methods like SVM, there is possibility to tune the regularization parameters. However we avoided extensive parameter tunings. In both MI logistic regression methods, we used a small, fixed regularization parameter $\lambda = 2$. In our MI AdaBoost algorithm, we restricted ourselves to 50 iterations to control the degrees of freedom. The base classifier used is an unpruned C4.5 decision tree, but *not* fully expanded (as explained above). Instead of using the default setting of minimal 2 instances per leaf, we set it to be 2 *bags* per leaf. More specifically, we used the average number of instances per bag shown in Table 4.2 as the size of one bag, and thus used 10 (instances) for Musk1 and 120 (instances) for Musk2.

As can be shown, the performance of our methods is comparable to some of the best methods in MI learning, and compares favorably to most of the "APR-like + MI assumption" family. More importantly, since the generative models and the assumptions of our methods are totally clear, it is easy to assess their applicability when new MI data becomes available. In the Musk datasets, it seems that logistic regression based on the arithmetic average of the instances' class probabilities performs better than based on the normalized geometric average, thus the assumptions of this method could be more realistic in this case. Finally, the overall performance of our methods is also comparable to that of the wrapper method described in Chapter 3, slightly worse in Musk2 data though. Although the methods in this chapter are the exact solutions to the underlying generative model, they do not seem superior to the heuristic methods with the biased probability estimates. This is especially true when the assumptions of the heuristic reasonably hold in reality, which seems to be the case in the Musk datasets.

4.6 Related Work

The Diverse Density (DD) algorithm [Maron, 1998] is the only probabilistic model for MI learning in the literature, and hence it is the work most closely related. DD also models $Pr(Y|B)$ and $Pr(Y|X)$, and [Maron, 1998] proposed two ways to model the relationship between them, namely the noisy-or model and the most-likely-cause model. Both ways were aimed to fit the MI assumption. The noisy-or model regards the process of determining a bag's class probability (i.e. $Pr(Y = 0, 1|B)$) as a Bernoulli process with n independent stages where n is the number of instances in the bag. In each stage, one decides the class label using the probability

$Pr(y|x)$ of an (unused) instance in the bag. Roughly speaking, a bag will be labeled positive if one sees at least one positive class label in this process and negative otherwise. The most-likely-cause model regards the above process as one-stage. It thus only picks one instance per bag. The selection of the instance is parametric or model-based, that is, according to the radial formulation of $Pr(Y|X)$ it uses the instance with the *maximal* probability $max_{x \in b}\{Pr(y = 1|x)\}$ in a *positive* example and the instance with the *minimal* probability $min_{x \in b}\{Pr(y = 0|x)\}$ in a *negative* example as the representative of the corresponding bag.

Indeed, one can recognize the similarity of our methods with DD, especially the linear logistic regression method — we simply take out the radial formulation and the noisy-or/most-likely-cause model, and plug in the linear logistic formulation and the geometric/arithmetic average model. It is natural to also try the "Gaussian-like" formulation together with the collective assumption (i.e. the geometric/arithmetic average model), or the "linear + MI assumption (i.e. noisy-or)" combinations. However these combinations do not work well on the Musk datasets according to our experiments. When experimented on the Musk1 dataset, for example, the "Gaussian-like + collective assumption (arithmetic average of probabilities)" combination has the 10×10-fold CV error rate of 19.02%±3.41%, and the "linear + MI assumption (noisy-or)" combination has 20.00%±1.47% with the ridge parameter set to $\lambda = 12$ (the high value of ridge parameter may already indicate the unsuitability of the linear model).

As a matter of fact, the whole family of "APR-like + MI Assumption"-based methods are related to this chapter because their rationale is the same as that of DD. The current methods that upgrade the single-instance learners (e.g. MI decision tree learner RELIC [Ruffo, 2001], MI nearest neighbour algorithms [Wang and Zucker, 2000], MI decision rule learner NaiveRipperMI [Chevaleyre and Zucker, 2001] and the SVM with an MI kernel and a polynomial minimax kernel [Gärtner et al., 2002]), on the other hand, are not so closely related to the framework presented here because the line of thinking is quite different. The SVM with the MI kernel [Gärtner et al., 2002] may also depend on the collective assumption because the MI kernel essentially uses every instance within a bag (with equal weights). Other methods' assumptions are not very clear because these methods are purely heuristic and therefore hard to analyze.

Finally we propose an improvement on DD. The radial (or Gaussian-like) formulation is not convenient for optimization because it has too many local maxima in the log-likelihood function. If one really believes the quadratic formulation is reasonable, we suggest a linear logistic model with polynomial fitting to order 2 (i.e. adding quadratic terms). Together with the noisy-or model (or the most-likely-cause model), this would enable us to search for local maxima of the log-likelihood only. If we were to use every possible term in a polynomial expansion, we would have too many parameters to estimate, especially for the Musk datasets in which we already have too many attributes in the original instance space. In order to overcome this difficulty, we can assume no interactions between any two attributes, which is effectively the same as DD did. By deleting any interaction terms in the quadratic expansion, we only have twice the number of attributes coefficients (a quadratic term and a linear term for each attribute) plus an intercept to estimate. The number of parameters is roughly the same as in DD. However this model will have three major improvements over DD:

1. The optimization is now a local optimization problem as mentioned before, which is much easier. Consequently the computational cost is greatly reduced. The time of running this model will be very similar to the MI logistic regression methods, which according to our observation is very short. We believe it is even faster than EM-DD regardless of EM-DD's validity (because EM-DD still needs many optimization trials with multiple starts) whereas we can ensure the correctness of the MLE solutions in this model;

2. The sigmoid function in the logistic model tends to make the log-likelihood function more gentle and smooth than the exponential function used in the radial model proposed by DD. As a matter of fact, the log-likelihood function in DD is discontinuous (the discontinuity occurs whenever the point variable's

value equals the attribute value of an instance in a negative bag) due to the radial form of the probability formulation. Besides, the exponential function in the probability formulation only allows DD to deal with data values around one, which means that we need to pre-process the data on most occasions. We can avoid this inconvenience in the logistic model. Despite the functional differences, the effects of the two models are virtually the same. To see why, in our model the log-odds function can always be arranged into a form that describes an Axis-Parallel hyper-Ellipse (APE) plus an intercept (bias) term, hence the instance-level decision boundaries of both models are exactly the same. But the gentleness of the sigmoid function in the logistic model (and the intercept term) may improve the prediction;

3. Since there are a lot of mature techniques that can be directly applied to the linear logistic model, like regularization or deviance-based feature selection, we can apply them directly to the new model. For example, ridge methods may further improve the performance if the instance-level decision boundary cannot be modeled well by an APE.

4.7 Conclusions

This chapter described a general framework for upgrading single-instance learners to deal with MI problems. Typically we require that the single-instance learners model the posterior probability $Pr(Y|X)$ or its transformation. Then we can construct a bag-level loss function with some assumptions. By minimizing the expected loss function at the bag level we can “recover” the instance-level probability function $Pr(Y|X)$. The prediction is based on the recovered function $Pr(Y|X)$ and the assumption used. This framework is quite general and justified thanks to the strong theoretical basis of most single-instance learners. It also incorporates background knowledge (based on the assumptions) involved and could serve as general-purpose guidance for solving MI problems.

Within this framework, we upgraded linear logistic regression and AdaBoost based the collective assumption. We have also shown that these methods, together with mild regularizations, perform quite well on the Musk benchmark datasets.

For "group-conditional" single-instance learners that estimate the density $Pr(X|Y)$ and then transform to $Pr(Y|X)$ via Bayes' rule (like naive Bayes or discriminant analysis), there appears to be no easy way to directly upgrade them. However we explore some methods in the next chapter that follow the same line of thinking and are very similar to the group-conditional methods.

Chapter 5

Learning with Two-Level Distributions

Multiple instance learning problems are commonly tackled in a point-conditional manner, i.e. the algorithms aim to directly model a function that, given a bag of n instances, $x_1, \cdots, x_n$, outputs a group (or class) label y. No methods developed so far build a model in a group-conditional manner. In this chapter we developed a group-conditional method, the "two-level distribution approach" to handle MI data. As the other approaches presented in this thesis, it essentially discards the standard MI assumption. However, in contrast to the other approaches, it derives a distributional property for each bag of instances. We describe its generative model and demonstrate that it is an asymptotically unbiased classifier based on artificial data. In spite of its simplicity, the empirical results of this approach on the Musk benchmark datasets are surprisingly good. Finally, we show the relationship of this approach with some single-instance group-conditional learners. We also discover its relationship with the empirical Bayes, an old statistical method, within a classification context, and the relationship of MI learning with meta-analysis.

5.1 Introduction

Many special-purpose MI algorithms can be found in the literature. However we observed that all these methods aim to directly model a function of $f(y|x_1, \cdots, x_n)$ where y is the group (class) label of a bag of n instances, and $x_1, \cdots, x_n$ are the instances. From a statistical point of view $f(.)$ is necessarily a probability function $Pr(.)$, although there can be other interpretations. We refer to this approach as a point-conditional approach. In normal single-instance learning, we have a category of popular methods that model group-conditional probability distributions and then transform to class probabilities using Bayes' rule. This category includes discriminant analysis [McLachlan, 1992], naive Bayes [John and Langley, 1995] and kernel density estimation [Hastie et al., 2001]. It is thus natural to ask whether it is possible to develop a group-conditional approach for MI algorithms.

In this chapter we present a group-conditional approach called "two-level distribution (TLD) approach". The underlying idea of this approach is simple: we extract distributional properties from each bag of instances for each class and try to discriminate classes (or groups) according to their distributional properties. Note that this approach essentially discards the standard MI assumption because there is no instance selection within each bag. Instead, by deriving the distributional properties of a bag we imply the collective assumption that was proposed earlier in this thesis.

This chapter is organized as follows. Section 5.2 describes the TLD approach in detail. It turns out that this approach is equivalent to extracting low-order sufficient statistics from each bag, which puts this approach into a broader framework. The underlying generative model of this approach is presented in Section 5.3, where we also show this approach is asymptotically unbiased if the generative model is true. In Section 5.4 we show the relationship of the TLD approach with the normal group-conditional single-instance learners. When assuming independence between attributes, one of our methods looks very similar to naive Bayes, thus we present an (approximate) upgrade of naive Bayes to MI learning. In Section 5.5 we show experimental results for the TLD approach on the Musk benchmark datasets. Section 5.6 draws the connection to the empirical Bayes (EB) methods in a classification context. It turns out that the TLD methods follow exactly the EB line of thinking. As an old statistical method, EB has many fielded applications in practice, especially the "meta-analysis" field in medical research. Such applications may draw interests to the MI domain. Finally Section 5.7 concludes this chapter.

5.2 The TLD Approach

First let us consider normal single-instance learning. The essence of the group-conditional methods is to derive distributional properties within each group (and the group priors) so that we can decide the frequency of a point in the instance space for each group. We classify a point according to the most frequently appearing group (class) label. We follow the same line of thinking in the multi-instance (MI) case. Dependent on different groups, we first derive distributional properties for each bag, that is, on the instance level. Because even within one class, the distributional properties are different from bag to bag, we need a second-level distribution to relate the instance-level distribution to one another. We refer to the second level as the "bag level" distribution. That is why we call this approach the "two-level distribution" (TLD) approach.

The obvious question is how to derive distributions over distributions? Since commonly used distributions are usually parameterized by some parameters, we can think of these parameters as random and governed by some hyper-distribution. This is essentially a Bayesian perspective [O'Hagan, 1994]. These hyper-distributions themselves are parameterized by some hyper-parameters. In a full Bayesian approach, we would cast further distributions on the hyper-parameters. In this chapter, we simply regard the hyper-parameters as fixed and assume that they can be estimated from the data. Therefore our task is to simply estimate the hyper-parameters for each group (class). We show the formal derivation of how to estimate them in the following.

First, we introduce some notation. We denote the j^{th} bags as b_j for brevity. Formally, if b_j has n_j instances, then $b_j = \{x_{j1}, \cdots, x_{jk}, \cdots, x_{jn_j}\}$. Y denotes the class variable. Then, given a class label $Y = y$ (in two-class case $y = 0, 1$) and a bag b_j, we have the distribution $Pr(b_j|Y)$ for each class, which is parameterized with a fixed bag-level parameter δ^y (hence we simply write down $Pr(b_j|Y)$ as $Pr(b_j|\delta^y)$). We estimate δ using the maximum likelihood method:

$$\delta^y_{MLE} = argmaxL = argmax \prod_j Pr(b_j|\delta^y).$$

Here L is the likelihood function, $\prod_j Pr(b_j|\delta^y)$. Now, the instances in bag b_j are not directly related to δ, as we discussed before. Instead, the instances x_{jk} are governed by an instance-level distribution parameterized by a parameter vector θ, that in turn is governed by a distribution parameterized by δ. Since θ is a random variable, we integrate it out in L. Mathematically,

$$\begin{aligned} L &= \prod_j Pr(b_j|\delta^y) \\ &= \prod_j \int Pr(b_j, \theta|\delta^y)\,\mathrm{d}\theta \\ &= \prod_j \int Pr(b_j|\theta, \delta^y)P(\theta|\delta^y)\,\mathrm{d}\theta \end{aligned}$$

and assuming conditional independence of b_j and δ^y given θ,

$$= \prod_j \int Pr(b_j|\theta)Pr(\theta|\delta^y)\,\mathrm{d}\theta. \tag{5.1}$$

In Equation 5.1, we effectively marginalize θ in order to relate the observations (i.e. instances) to the bag-level parameter δ. Now assuming the instances within a bag are independent and identically distributed (i.i.d) according to a distribution parameterized by θ, $Pr(b_j|\theta) = \prod_i^{n_j} P(x_{ji}|\theta)$ where x_{ji} denotes the i^{th} instance in the j^{th} bag.

The complexity of the calculus would have stopped us here had we not assumed independence between attributes. Thus, like naive Bayes, we assume independent attributes, and reduce the integral in Equation 5.1 into several one-dimensional integrals (one for each attribute), which are much easier to solve. Now, assuming m dimensions and e bags (for one class), the likelihood function L becomes

$$\begin{aligned}
L &= \prod_{j=1}^{e} \iint \cdots \iint \Bigg\{ \Big[\prod_{i=1}^{n_j} Pr(x_{jki}|\theta_k) \Big] Pr(\theta_k|\delta_k^y) \Bigg\} \, \mathrm{d}\theta_1 \cdots \, \mathrm{d}\theta_m \\
&= \prod_{j=1}^{e} \Big(\prod_{k=1}^{m} \Bigg\{ \int \Big[\prod_{i=1}^{n_j} Pr(x_{jki}|\theta_k) \Big] Pr(\theta_k|\delta_k^y) \, \mathrm{d}\theta_k \Bigg\} \Big) \\
&= \prod_{j=1}^{e} \Big(\prod_{k=1}^{m} B_{jk} \Big) \qquad (5.2)
\end{aligned}$$

where x_{jki} denotes the value of the k^{th} dimension of the i^{th} instance in the j^{th} exemplar, θ_k and δ_k are the parameters for the k^{th} dimension, and

$$B_{jk} = \int \Big[\prod_{i=1}^{n_j} Pr(x_{jki}|\theta_k) \Big] Pr(\theta_k|\delta_k^y) \, \mathrm{d}\theta_k .$$

There are many options for modeling $Pr(x_{jki}|\theta_k)$ and $Pr(\theta_k|\delta_k^y)$. Here we model $Pr(x_{jki}|\theta_k)$ as a Gaussian with parameters μ_k and σ_k^2:

$$\begin{aligned}
\prod_{i=1}^{n_j} Pr(x_{jki}|\theta_k) &= \prod_{i=1}^{n_j} Pr(x_{jki}|\mu_k, \sigma_k^2) \\
&= (2\pi\sigma_k^2)^{-n_j/2} \exp \Big[- \frac{S_{jk}^2 + n_j(\overline{x}_{jk} - \mu_k)^2}{2\sigma^2} \Big] \qquad (5.3)
\end{aligned}$$

where $\overline{x}_{jk} = \sum_{i=1}^{n_j} x_{jki}/n_j$ and $S_{jk}^2 = \sum_{i=1}^{n_j}(x_{jki} - \overline{x}_{jk})^2$. As is usually done in Bayesian statistics [O'Hagan, 1994], we conveniently model $Pr(\theta_k|\delta_k^y)$ as the corresponding *natural conjugate* form of the Gaussian distribution. The natural conjugate has four parameters, a_k, b_k, w_k, and m_k, and is given by:

$$Pr(\theta_k|\delta_k^y) = g(a_k, b_k, w_k)(\sigma_k^2)^{-\frac{b_k+3}{2}} \exp \Big(- \frac{a_k + \frac{(\mu_k - m_k)^2}{w_k}}{2\sigma_k^2} \Big) \qquad (5.4)$$

where

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

Taking a closer look at the natural conjugate prior in Equation 5.4, it is straightforward to see that μ_k follows a normal distribution with mean m_k and variance $w_k\sigma_k^2$, and $(\frac{a_k}{\sigma_k^2})$ follows a Chi-squared distribution with b_k degrees of freedom (d.f.). It is Chi-squared because σ_k^2 is positive, and so is $(\frac{a_k}{\sigma_k^2})$ given that $a_k > 0$. In the Bayesian literature [O'Hagan, 1994], σ_k^2 is said to follow an *Inverse-Gamma* distribution. The natural conjugate prior is quite reasonable, but the fact that the variance of μ_k is a multiple (w_k) of σ_k^2 is unrealistic and uninterpretable, although it brings a considerable convenience to the calculus (In fact we could not find a simple analytical form of the likelihood function without this dependency). We will drop this dependency later on when we simplify the model. Plugging Equation 5.3 and Equation 5.4 into the middle part of the likelihood function in Equation 5.2 we get B_{jk}:

$$\begin{aligned} B_{jk} &= \int \Big[\prod_{i=1}^{n_j} Pr(x_{jki}|\theta_k)\Big] Pr(\theta_k|\delta_k^y)\ \mathrm{d}\theta_k \\ &= \int_0^{+\infty}\int_{-\infty}^{+\infty}\Bigg\{(2\pi\sigma_k^2)^{-n_j/2}\exp\Big[-\frac{S_{jk}^2 + n_j(\overline{x}_{jk}-\mu_k)^2}{2\sigma^2}\Big] \\ &\qquad g(a_k, b_k, w_k)(\sigma_k^2)^{-\frac{b_k+3}{2}}\exp\Big[-\frac{a_k + \frac{(\mu_k-m_k)^2}{w_k}}{2\sigma_k^2}\Big]\Bigg\}\ \mathrm{d}\mu_k\ \mathrm{d}\sigma_k^2 \end{aligned} \tag{5.5}$$

The integration is easy to do due to the form of the natural conjugate prior, resulting in (the details of the calculation are given in Appendix C):

$$= \frac{a_k^{b_k/2}(1+n_jw_k)^{(b_k+n_j-1)/2}\Gamma(\frac{b_k+n_j}{2})}{\Big[(1+n_jw_k)(a_k+S_{jk}^2)+n_j(\overline{x}_{jk}-m_k)^2\Big]^{\frac{b_k+n_j}{2}}\pi^{\frac{n_j}{2}}\Gamma(\frac{b_k}{2})} \tag{5.6}$$

Thus, the log-likelihood is:

$$LL = \log[\prod_{j=1}^{e}(\prod_{k=1}^{m} B_{jk})] = \sum_{k=1}^{m}\sum_{j=1}^{e}(-\log B_{jk}) \tag{5.7}$$

where B_{jk} is given in Equation 5.6 above. We maximize the log-likelihood LL

in Equation 5.7 using a constrained optimization procedure, to get δ^y_{MLE}, four parameters per attribute. Note that LL actually only involves the sample mean $\overline{x}_{jk}$ and sum of square errors S_{jk}, thus it turns out that this method extracts low-order sufficient statistics, which is a kind of metadata, from each bag to estimate the group-conditional hyper-parameters.

When a new bag is encountered, we simply extract these statistics from the bag. We then compute the log-odds:

$$\log \frac{Pr(Y=1|b_{test})}{Pr(Y=0|b_{test})} = \log \frac{Pr(b_{test}|\delta^1_{MLE})Pr(Y=1)}{Pr(b_{test}|\delta^0_{MLE})Pr(Y=0)} \tag{5.8}$$

We estimate the prior $Pr(Y)$ from the number of bags for each class in the training data and classify b_{test} according to whether the log-odds value is greater than zero.

In spite of its sound theoretic basis, the above method does not perform well on the Musk datasets, mainly due to the unrealistic dependency of the variance of μ_k on σ^2_k in the natural conjugate, as well as the restrictive assumptions of an Inverse-Gamma for σ^2 and a Gaussian for the instances within a bag. However regarding this as a metadata-based approach allows us to simplify it in order to drop some assumptions. Here we make two simplifications.

The first simplification stems from the Equation 5.3. It is relatively straightforward to see that the likelihood within each bag is equivalent to the product of the sampling distributions of the two statistics $\overline{x}_{jk}$ and S_{jk}.[1]. If we think $\overline{x}_{jk}$ is enough for the classification task, we can simply drop the second-order statistics S_{jk} here. By dropping S_{jk} we can generalize this method to virtually any distribution within a bag because according to the central limit theorem, $\overline{x}_{jk} \sim N(\mu_k, \frac{\sigma^2_k}{n_j})$ no matter how the x_{jki}'s are distributed. The second simplification is that we no longer model σ^2_k as drawn from an Inverse-Gamma distribution in Equation 5.4. Instead we regard it as fixed and directly estimate $\hat{\sigma^2_k}$ from the data. Accordingly we do not need to estimate

[1] For a Gaussian distribution, the sampling distributions are $\overline{x}_{jk} \sim N(\mu_k, \frac{\sigma^2_k}{n_j})$ and $\frac{S_{jk}}{\sigma^2_k} \sim \chi^2(n_j - 1)$ If we multiply these two sampling distributions, we will get the same form as in Equation 5.3, differing only by a constant.

a_k and b_k any more. To estimate σ_k^2, we give a weight of $\frac{1}{n}$ to each instance where n is the number of instances in the corresponding bag (because intuitively we regard a bag as an object and thus should assign each bag the same weight). Based on the weighted data, an unbiased estimate of σ_k^2 is $\hat{\sigma_k^2} = \sum_i [\frac{1}{n_j} \sum_i (x_{jki} - \overline{x}_{jk})^2]/(e - \sum_j \frac{1}{n_j})$, where e is the number of bags.

With these two simplifications, the dependency in the natural conjugate prior disappears. We only have a Gaussian-Gaussian model within the integral in Equation 5.1, which makes the calculus extremely simple. The resulting formula is again a Gaussian, which is

$$B_{jk} = \left(2\pi \frac{w_k n_j + \sigma_k^2}{n_j}\right)^{-1/2} \exp\left[\frac{-n_j(\overline{x}_{jk} - m_k)^2}{2(w_k n_j + \sigma_k^2)}\right] \tag{5.9}$$

Note that w_k in the first TLD method denotes the ratio of the variance of μ_k to σ_k^2. Here, since we have dropped this dependency, w_k is simply the variance of μ_k. Equation 5.9 means that $\overline{x}_{jk} \sim N(m_k, w_k + \frac{\sigma_k^2}{n_j})$. Thus it basically tells us that the means of the bags of the two classes are wandering around two Gaussian centroids respectively, although with different variances, as shown in Figure 5.1. We simply substitute Equation 5.6 in the log-likelihood function by Equation 5.9. The MLE of m_k is in a simple analytical form but w_k is not. Thus we still need a numeric optimization procedure to search for a solution. However, since the search space is two-dimensional (for convenience, we also search for $\hat{m_k}$ even though its solution can be obtained directly) and has no local maxima, the search is very fast. This constitutes our simplified TLD method and we call it "TLDSimple", while we call the former TLD method simply "TLD".

5.3 The Underlying Generative Model

The underlying generative model of the TLD approach is straightforward — we simply have distributions on two levels, and it is also straightforward to generate data from it. First, we generate some random data from the distribution parameterized by the fixed bag-level hyper-parameters. Then we regard this data as the instance-level parameters and generate instance-level random data according to some distribution parameterized by these instance-level parameters.

We can generate artificial data based on this generative model for both TLD and TLDSimple. The artificial data generated using the generative model assumed by the simplified TLD method is easy to visualize. An example is in Figure 5.1. More specifically, we generated data from two levels of Gaussians along two independent dimensions for two classes (black and grey in the graph). First, we identified two (bag-level) Gaussians, one for each class, with the same variance but different means. We then generated 10 data points from each Gaussian and regard each of the points as the mean of an instance-level Gaussian. Finally we specified a fixed variance (same for both classes) for the instance-level Gaussians and generated 1 to 20 data points from each instance-level Gaussian to form a bag. The number of instances per bag is uniformly distributed.

In Figure 5.1 we plot the contour of both levels of Gaussians with 3 standard deviations. The dot inside each Gaussian is the mean and we can observe its variance from the dispersion along each dimension. The straight line between two bag-level Gaussians is the *bag-level* decision boundary. Note that unlike the methods that intend to “recover” the instance-based decision pattern such as the APR algorithms [Dietterich et al., 1997], Diverse Density [Maron, 1998] or other methods developed in this thesis, the instance-level decision boundary is not defined here. Instead only the decision boundary of the true means (i.e. μ) of the bags is well-defined — it is a hyperplane because we used the same variance for both classes[2]. Also note that the decision boundary is only defined in terms of the true parameters μ, but *not* even in terms of the statistics (or metadata) $\overline{x}$. This is due to the extra term $(\frac{\sigma^2}{n_j})$ in the variance of $\overline{x}$. This term is distinct for each bag even if we have the same σ^2 but different number of instances per bag.

We generated the artificial data mainly to analyze the properties of our methods.

[2] If the Gaussian of either class had different variance, then the decision boundary would be quadratic instead of linear.

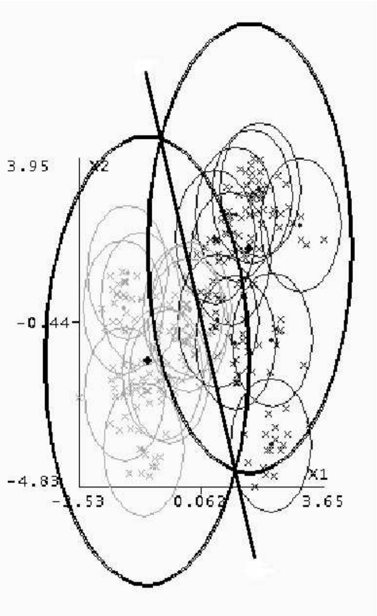


Figure 5.1: An artificial simplified TLD dataset with 20 bags.

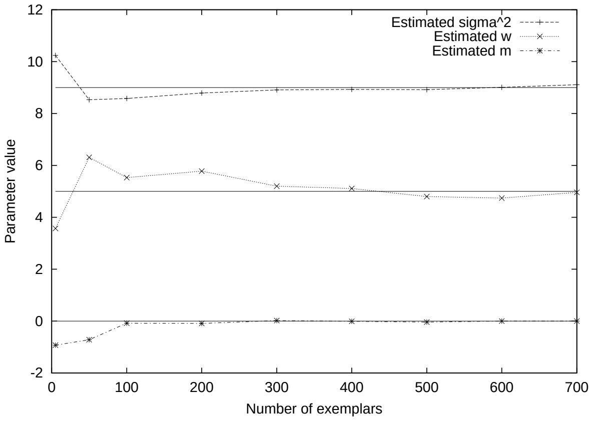
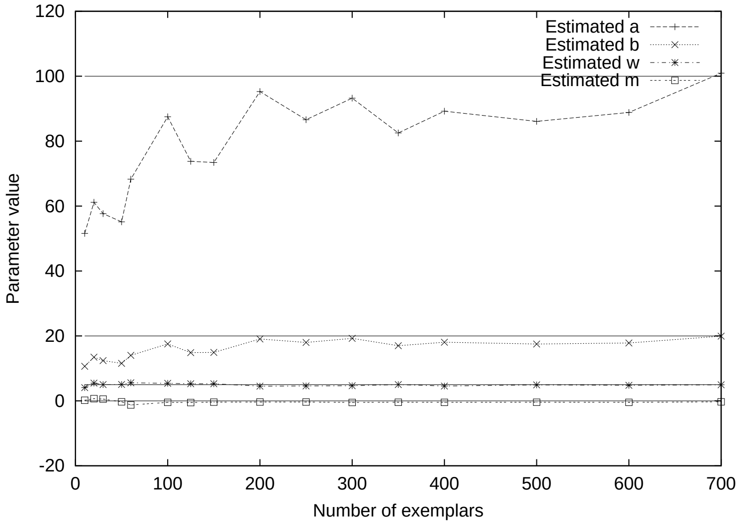


Figure 5.2: Estimated parameters using the TLDSimple method.

In order to verify the (at least asymptotic) unbiasedness of both TLD methods, we generated a one-dimensional dataset with 20 instances per bag using the exact generative model assumed by them (the data generation using the exact generative model assumed by TLD is described in Chapter 7 and Appendix A). The estimated parameters (of one class) are shown in Figure 5.2 for TLDSimple and Figure 5.3 for TLD. In both figures, the solid lines denote the true parameter values. Note that in TLDSimple, $\hat{\sigma}_k^2 = \frac{\sum_i[\frac{1}{n_j}\sum_j(x_{jki}-\overline{x}_{jk})^2]}{(e-\sum_j\frac{1}{n_j})}$ is not a maximum likelihood estimate (MLE), but obviously it is an unbiased estimate as mentioned in Section 5.2 because

$$E\Big(\sum_j[\frac{1}{n_j}\sum_i(x_{jki}-\overline{x}_{jk})^2]\Big) = (e-\sum_j\frac{1}{n_j})\sigma_k^2.$$

All other estimates are MLEs. As can be seen, the estimated parameters converge to the true parameters as the number of bags increases, thus this method is at least asymptotically unbiased. We observe that the number of instances per bag does not affect the convergence but only the rate of convergence. The more instances per bag, the faster the convergence. Varying number of instances will slow down the convergence but result in similar behaviors.



10cm

Figure 5.3: Estimated parameters using the TLD method.

In TLDSimple, the MLE of the variance of μ (i.e. w) is a biased one, but still asymptotically unbiased. The reason why we use the MLE instead of another estimate is due to its robustness. We have observed that when the number of bags is small, which is the case in the Musk datasets, other estimates of the variance are very likely to be wildly wrong, even negative. The MLE, on the other hand, is relatively stable even for a small sample size.

5.4 Relationship to Single-instance Learners

The relationship between the TLD approach and single-instance learners can best be explained based on the TLDSimple method. Note that when the number of instances per bag is reduced to 1, Equation 5.9 degrades into the same Gaussian for all bags (or instances in this case) from one class, with mean m_k and variance w_k, because σ_k^2 cannot be estimated and can only be regarded as 0. Hence this method degrades into naive Bayes in the single-instance case. If we had dropped the independence assumption between attributes, we would have obtained the discriminant analysis method. Even in the multi-instance case, if $\sigma_k^2 \approx 0$ and/or $n_j \gg 0$, the term $\frac{\sigma_k^2}{n_j}$ can be neglected in Equation 5.9. In that case for all the bags in one class, the $\overline{x}_{jk}$'s can be regarded as coming from the same Gaussian. Then we can use standard naive Bayes to simulate the TLDSimple method — we simply calculate the sample average over every bag along all dimensions and construct one instance per bag using the sample average. Then we use naive Bayes to learn this single-instance data. This gives an approximate upgrade of naive Bayes in the MI case. Such an approximation may be appropriate in the Musk datasets because we observed that the in-bag variance is indeed very small for many bags.

The above view of the TLD approach leads us to consider improvement techniques from naive Bayes in the TLDSimple method. In TLDSimple, it is assumed that the μ_k's are normally distributed (i.e. $\sim N(m_k, w_k)$). But if the normality does not hold very well, we need some adjustment techniques. We present one simple technique here. When a new bag b_{test} is met, we calculate the log-odds function according to Equation 5.8 and usually decide its class label based on whether the log-odds $>$ 0 or not. Now, we determine the class label of b_{test} based on whether the log-odds $> v$ or not, where v is a real number. We choose v so as to minimize the classification errors in the training data. We call v the "cut-point". Thus we select an empirically optimal cut-point value instead of 0. This technique was mentioned in the context of discriminant analysis [Hastie et al., 2001] but it is obviously applicable in our TLD approach and in naive Bayes.

The rationale of this technique is easy to see, and illustrated in Figure 5.4 in one dimension. Here we have one Gaussian and one standard Gamma distribution for each class (solid lines). The dotted line plots the Gaussian estimated using the mean and variance of the Gamma. Now the estimated decision boundary becomes C' whereas it should be C. Since in classification we are only concerned about the decision boundary, we do not need to adjust the density. By looking for the empirically optimal cut-point, we can move it from C' back to C and improve prediction performance.

More specifically, we look for the cut-point as follows. First, we calculate the log-odds ratio for each of the training bags and sort them in ascending order. The log-odds ratios of bags with class 1 should be greater than those of bags with class

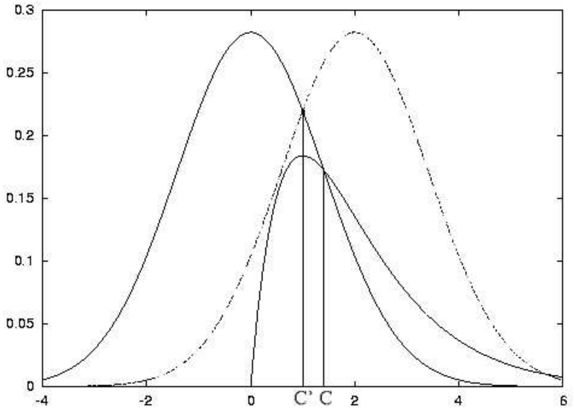


Figure 5.4: An illustration of the rationale for the empirical cut-point technique.

0. Second, we look for a split. Let T denotes the total number of training bags, and S a split point. There will be t bags with log-odds $\leq S$ and $(T-t)$ ones with log-odds $> S$. We count the number of bags in the first class among the first t bags. This number is p_1. The number of bags in the second class among the second set of $(T-t)$ bags is p_2. Finally, we go through the log-odds of all bags from the smallest to the largest and find the log-odds value with the largest value of (p_1+p_2) and use that value as the cut-point. If there is a tie, we choose the value closest to 0.

This technique is not commonly used to reduce the potential negative impact of the normality assumption. Instead kernel density estimation or discretization of numeric attributes are more often used. Nonetheless, we found that this technique can also work pretty well in the normal single-instance learning. We have tested it in association with naive Bayes on some two-class datasets from the UCI repository [Blake and Merz, 1998] within the experimental environment of the WEKA workbench [Witten and Frank, 1999]. The results are shown in Table 5.1.

In Table 5.1 we use “NB” to denote naive Bayes. The first column is NB with empirical cut-point (EC) selection and this is the base line for comparison. The second, third and fourth columns are NB without any adjustment, NB with kernel density estimation (KDE) and NB with disretization of numeric attributes (DISC) respectively. The results were obtained using 100 runs of 10-fold cross validation

Dataset	**NB+EC**	**NB**	**NB+KDE**	**NB+DISC**
breast-cancer	72.53(0.96)	72.76(0.68)	72.76(0.68)	72.76(0.68)
breast-cancer-W	95.87(0.23)	96.07(0.1)	97.51(0.11) v	97.17(0.13) v
german-credit	74.75(0.51)	75.07(0.43)	74.5 (0.46)	74.38(0.62)
heart-disease-C	82.58(0.75)	83.4 (0.41)	84.02(0.58) v	83.2 (0.64)
heart-disease-H	83.52(0.71)	84.23(0.52)	84.95(0.39) v	84.12(0.36)
heart-statlog	83.78(0.74)	83.73(0.61)	84.4 (0.59)	82.91(0.67)
hepatitis	83.33(1)	83.71(0.82)	84.76(0.62) v	83.67(1.14)
ionosphere	89.68(0.54)	82.51(0.45) *	91.83(0.32) v	89.27(0.45)
kr-vs-kp	87.85(0.16)	87.8 (0.15)	87.8 (0.15)	87.8 (0.15)
labor	93.29(2.16)	94 (1.92)	93.18(1.36)	88.44(1.85) *
mushroom	98.18(0.03)	95.76(0.04) *	95.76(0.04) *	95.76(0.04) *
sick	95.99(0.09)	92.76(0.12) *	95.78(0.08) *	97.14(0.08) v
sonar	72.08(1.39)	67.88(1.06) *	72.68(1.13)	76.43(1.51) v
vote	89.55(0.63)	90.09(0.15)	90.09(0.15)	90.09(0.15)
pima-diabetes	75.43(0.46)	75.65(0.37)	75.15(0.37)	75.51(0.74)
Summary	(v/ /*)	(0/11/4)	(5/8/2)	(3/10/2)

Table 5.1: Performance of different versions of naive Bayes on some two-class datasets.

(CV), with standard deviations specified in brackets. The confidence level used for the comparison was 99.5%. We use a "*" sign to indicate "significantly worse than" where as "v" means "significantly better than". We regard two results of a dataset as "significant different" if the difference is statistically significant at the 99.5% confidence level according to the corrected resampled t-test [Nadeau and Bengio, 1999]. It can be seen that the EC technique is comparable to discretization and worse than KDE in general. However, it can indeed improve the performance of the original naive Bayes. Moreover, in cases where KDE cannot be used, like in our TLD approach, EC can be a convenient option.

It turns out that in the Musk datasets the normality assumption for μ_k is a problem and the EC technique can apply. For instance, for the naive Bayes approximation of TLDSimple without EC, the leave-one-out (LOO) error rates on the Musk1 and Musk2 datasets are 13.04% and 17.65% respectively. With EC, the error rates are 10.87% and 14.71%, which is an improvement of about 3% in each dataset. Note that in the naive Bayes approximation of TLDSimple we do not use KDE because

Methods	**Musk 1**		**Musk 2**	
	LOO	**10CV**	**LOO**	**10CV**
SVM with Minimax Kernel	7.6	8.4±0.7	13.7	13.7±1.2
RELIC	–	16.3	–	12.7
TLDSimple+EC	14.13	16.96±1.86	9.80	15.88±2.56
naive Bayes Approximation+EC	10.87	15.11±2.32	14.71	17.35±1.85

Table 5.2: Error rate estimates from 10 runs of stratified 10-fold CV and LOO evaluation. The standard deviation of the estimates (if available) is also shown.

we do not want to accurately estimate the density of $\overline{x}_{jk}$ but that of μ_k. In this approximation, we ignored the term $\frac{\sigma_k^2}{n_j}$ in the variance of $\overline{x}_{jk}$. Hence $\overline{x}_{jk}$ is only an approximation of μ_k. The EC technique is only concerned about the cut-point instead of the whole density, and thus does not suffer from this approximation.

5.5 Experimental Results

There are no other directly related MI methods in the literature. The closest cousins may be other methods that also extract metadata from bags, which are the MI decision tree learner RELIC [Ruffo, 2001] and the support vector machine (SVM) with the minimax kernel [Gärtner et al., 2002]. Both methods can be viewed as extracting minimax metadata from each bag and applying a standard learner to the resulting single-instance data. Although it has been mentioned that some statistics can be extracted from bags for the purpose of modeling [Gärtner et al., 2002], no specific methods have been put forward in the MI learning domain.

We present experimental results for TLDSimple and its naive Bayes approximation (both used with the EC technique) on the Musk benchmark datasets in Table 5.2. We also present published results for RELIC and the SVM method in the table. Since these two metadata-based methods are among the best methods on the Musk datasets, we can see that the new methods are competitive with the state-of-the-art MI methods.

In spite of its simplicity, TLDSimple performs surprisingly well on the Musk2 dataset, with reasonably good result on Musk1. Since it is a distributional method, it is expected that more data can improve its estimation and that's why its LOO performance is better than that of 10-fold CV. Its naive Bayes approximation is equally simple and also quite good on the datasets. Hence the sound theoretical basis of these methods appears to bear fruits in practical datasets.

5.6 Related Work

As mentioned before, RELIC and the SVM with a minimax kernels are related to the TLD approach in the sense that they both extract metadata from the bags and model directly on the bag-level. But there are no other multi-instance methods that model the MI problem in a group-conditional manner, neither are there any methods trying to extract low-order sufficient statistics. Therefore our approach is quite unique in the MI learning domain.

However, when developing the TLD methods, we discovered empirical Bayes (EB) [Maritz and Lwin, 1989], an old statistical method that has the exactly same line of thinking as our approach. As a matter of fact, it is well suited for multi-instance data and our approach is one of its applications to classification. In the EB literature a more general EM-based solution procedure is also proposed that can model arbitrary distributions on the two levels. In such a case, the integral may not be solvable but — via the EM algorithm — it is possible to apply numeric integration techniques to make the maximum likelihood method feasible. This is actually one of the early applications of the EM algorithm [Dempster et al., 1977].

According to the EB literature [Maritz and Lwin, 1989], an early example of an EB application was an experiment with contaminated water [von Mises, 1943]. In this experiment, 3420 batches of water were collected with five samples in each batch. Each sample was either classified as positive (contaminated) or negative (uncontaminated). The task was to estimate the probability of a sample to be positive. One may recognize this dataset as very similar to MI data. If we classified each batch as positive or negative (according to some assumptions), and had some attributes to describe the samples, we would end up with an MI dataset.

Today EB is an important method to solve the "meta-analysis" problem in medical (or other scientific) research, which has a similar rationale as MI learning. In scientific research, different scientists carry out experiments associated with the same topic but usually get different results. If we can construct some instances (with the same attributes) for each one of the experiments, and try to identify the homogeneity and heterogeneity of these experiments, this is actually an MI dataset. Each experiment is a bag of instances but there is only one class label per bag. MI learning solutions will be very useful here because the identification of homogeneity/heterogeneity of scientific experiments is an important objective in meta-analysis.

We are presenting the relationship of the EB method and meta-analysis with MI learning in the hope that they may attract some interest in the field of MI research. We believe that this may help spawn fielded applications and more publicly available datasets — the perhaps most severe obstacles in research on MI learning today.

5.7 Conclusions

In this chapter we have proposed a two-level distribution (TLD) approach for solving multiple instance (MI) problems. We built a model in a group (class)-conditional manner, and think of the data as being generated by two levels of distributions — the instance-level distribution and the bag-level distribution — within each group. It turns out that the modeling process can be based on low-order sufficient statistics, thus we can regard it as one of the methods that are based on meta-data extracted from each bag.

Despite its simplicity, we have shown that a simple version of TLD learning and its naive Bayes approximation perform quite well on the Musk benchmark datasets. We have also showed the relationship of our approach with normal group-conditional single-instance learners, other metadata-based MI methods, and the empirical Bayes method. Finally, we pointed out the similarity between MI learning problems and the meta-analysis problems from scientific research.

Chapter 6

Applications and Experiments

As mentioned in Chapter 1, we focus on three practical applications of MI learning: drug activity prediction, fruit disease prediction, and content-based image categorization. Throughout this chapter, the term “number of attributes” does not include the “Bag-ID” attribute and the class attribute in the datasets. When we refer to a “positive” bag in this chapter, we simply mean that its class label is “1”. Likewise “negative” for “0” in the data.

6.1 Drug Activity Prediction

The most famous application for MI learning is the drug activity prediction problem, first described in [Dietterich et al., 1997], which resulted in the first MI datasets, the Musk datasets. These are the only real-world MI datasets publicly available. While the interested readers should refer to the paper of [Dietterich et al., 1997] for detailed background on the Musk datasets, we briefly describe them in Section 6.1.1. Section 6.1.2 we discuss another molecule activity prediction problem — predicting the mutagenicity of the molecules. Note that this problem was not originally an MI problem. However, with proper transformations we can use its data in MI learning.

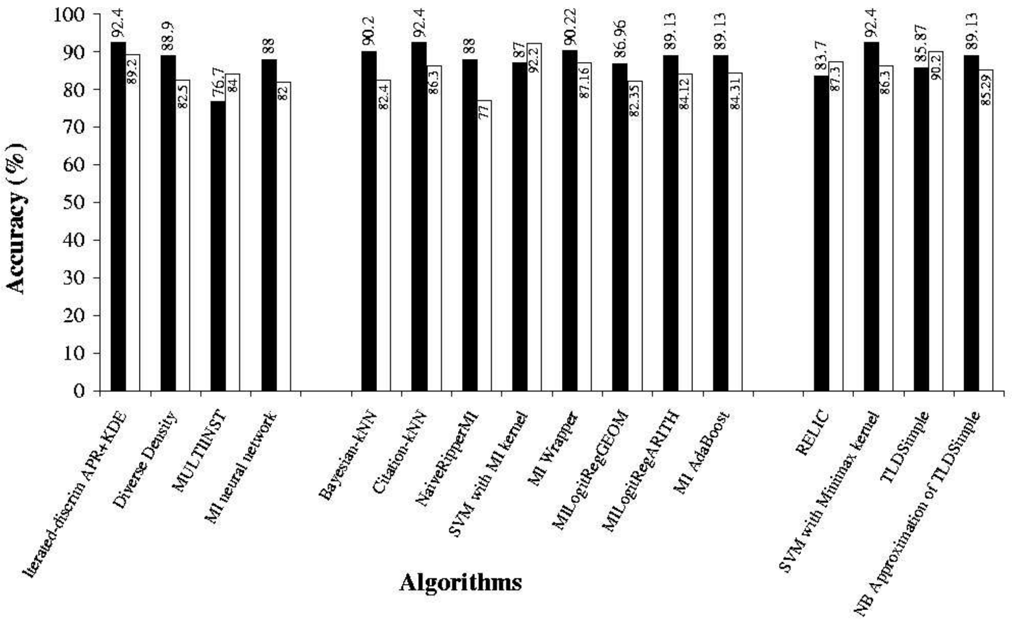


Figure 6.1: Accuracies achieved by MI Algorithms on the Musk datasets.

6.1.1 The Musk Prediction Problem

There are two Musk datasets, each describing different molecules (some molecules are presented in both of them). The task is to decide which molecules result in a “musky” smell. On average, Musk2 has more bags than Musk1, and each bag has more instances. Table 4.2 in Chapter 4 lists some key properties of the two datasets.

In both datasets, each molecule is represented as a bag and different shapes (or conformations) of that molecule are instances in the bag. Since one molecule has multiple shapes, we have to use multiple instances to represent an example. The shape of a molecule is measured using a ray-based representation that results in 162 features. Together with 4 extra oxygen features, we have 166 attributes. Since every shape of a molecule (i.e. every instance) can be measured with these 166 attributes, we can construct an MI dataset. Both Musk datasets have the same attributes as shown in Table 4.2 in Chapter 4.

The Musk datasets are the most popular ones in the MI domain. Virtually every MI algorithm developed so far has been tested on this problem and so have the algorithms in this thesis. Although we have used these datasets throughout this thesis to compare the performance of our methods to that of many other MI methods, we list the accuracies again in Figure 6.1. Based on this figure we can see some evidence which assumption really holds in the Musk datasets by examining the accuracies of methods based on various assumptions. We use black bars to denote the accuracy on Musk1, and white ones for Musk2. We used the best accuracy an algorithm can achieve, regardless of the evaluation method used. The accuracy of the wrapper method developed in Chapter 3 ("MI Wrapper" in the figure) is that achieved by wrapping around bagged PART with discretization. The MI AdaBoost method shown in the figure has unpruned regularized C4.5 as the base classifier and uses 50 iterations, same as described in Chapter 4. The two MI linear logistic regression methods ("MILogitRegGEOM" and "MILogitRegARITH" in the figure) are also described in Chapter 4. TLDSimple and the naive Bayes (NB) approximation of TLDSimple are described in Chapter 5.

We deliberately divide the algorithms into 3 categories: the first corresponds to instance-based methods that are consistent with the MI assumption; the second corresponds to the instance-based methods that we believe do not use the MI assumption; and the last corresponds to the metadata-based methods that cannot possibly use the MI assumption. As the figure shows, the methods that strongly depend on the MI assumption do not seem to benefit much from this background knowledge. Instead the overall performance of the methods that are not based on this assumption is as good as the "MI assumption" family on these datasets. Therefore the validity of the MI assumption as the background knowledge on the Musk datasets appears to be questionable.

6.1.2 The Mutagenicity Prediction Problem

There is another drug activity prediction problem, namely the mutagenicity prediction problem [Srinivasan, Muggleton, King and Sternberg, 1994], that can also be represented as an MI problem. It originated from the inductive logic programming

	Friendly	**Unfriendly**
Number of bags	188	42
Number of attributes	7	7
Number of instances	10486	2132
Number of positive bags	125	13
Number of negative bags	63	29
Average bag size	55.78	50.76
Median bag size	56	46
Minimum bag size	28	26
Maximum bag size	88	86

Table 6.1: Properties of the Mutagenesis datasets.

(ILP) domain. The original dataset is in a relational format, which is especially suitable for ILP algorithms. However we can transform the original dataset into a MI dataset in several ways [Chevaleyre and Zucker, 2000; Weidmann, 2003]. Here we consider a setting representing each molecule as a set of bonds and the pairs of atoms connected by each of the bonds. After this transformation each molecule corresponds to a bag. But unlike the Musk datasets where an instance describes the conformation of an entire molecule, each instance denotes an atom-bond-atom fragment of a molecule. One such fragment is described using 7 attributes — 5 symbolic and 2 numeric attributes. Interested readers may refer to [Weidmann, 2003] for more details on the construction of these MI datasets. The same construction method was used in other studies of MI learning [Chevaleyre and Zucker, 2001; Gärtner et al., 2002].

The original Mutagenesis dataset consists of 230 molecules: 188 molecules that can be fitted using linear regression and the remaining 42, which are more difficult to fit. These two subsets are called “friendly” and “unfriendly” respectively. The key properties of the two datasets, after the transformation, are shown in Table 6.1.

We evaluated some of the methods developed in this thesis on these two datasets. Since the TLD approach can only deal with numeric attributes and there are some nominal attributes in the Mutagenesis datasets, we cannot apply TLD methods to them. We normally consider the transformation of nominal attributes to binary ones

Method	**Friendly**	**Unfriendly**
Best of the TLC methods	9.31±1.28	18.33±1.61
MILogisticRegressionGEOM	15.74±0.91	16.67±0
MI AdaBoost	13.62±1.31	18.57±2.19
Wrapper with Bagged PART	18.78±1.26	23.10±2.26
Best of ILP methods	18.0	—
SVM with RBF MI kernel	7.0	25.0

Table 6.2: Error rate estimates for the Mutagenesis datasets and standard deviations (if available).

in this case. However the TLD methods involves estimating in-bag variance and this transformation often introduces zero variances for many attributes. Consequently the TLD approach cannot be applied even after such transformation. Hence we only applied the instance-based methods, i.e. the wrapper method, MI linear logistic regression methods and MI AdaBoost. In contrast to the experiments with the Musk datasets, MILogisticRegressionGEOM outperforms MILogisticRegressionARITH and thus we only present the results for MILogisticRegressionGEOM. Note that logistic regression cannot deal with nominal attributes either, but it can deal with the transformed binary attributes. Hence we transform a nominal attributes with K values into K binary attributes, each with value 0 and 1. As for the regularization, we (again) use a fixed ridge parameter λ instead of CV. Here we use $\lambda = 0.05$ for the "friendly" data and $\lambda = 2$ for the "unfriendly" data. MI AdaBoost, on the other hand, can naturally deal with nominal attributes, and we apply it directly to the Mutagenesis datasets. For the regularization in MI AdaBoost, we use the same strategy as in the Musk datasets. The base classifier is an unpruned regularized C4.5 decision tree [Quinlan, 1993] as in Chapter 4. The minimal number of leaf-node instances was set to around twice the averaged bag size, i.e. 120 instances for "Friendly" and 100 instances for "Unfriendly". Under these regularization conditions, we then look for a reasonable number of iterations. It turns out that the best performance often occurs with 1 iteration on the "unfriendly" data, which lead us to think that maybe decision trees are too strong for this dataset, and we should use decision stumps instead. Hence we eventually used 200 iterations of regularized C4.5 on the "friendly" dataset and 250 iterations of decision stumps on the "unfriendly" dataset. The estimated errors over 10 runs of 10-fold CV are shown in Table 6.2. Table 6.2 also shows the error estimates of the wrapper method described in Chapter 3 on these two datasets. The classifier used in the wrapper was Bagging [Breiman, 1996] based on the decision rule learner PART [Frank and Witten, 1998], both implemented in WEKA [Witten and Frank, 1999]. Bagged PART will also be used for the experiments with the wrapper method on other applications in this chapter. We used the default parameter setting of the implementations in WEKA, and did not do any further parameter tuning.

It was reported that the ILP learners have achieved error rates ranging from 39.0% (FOIL) to 18.0% (RipperMI) on the "friendly" dataset using one run of 10-fold CV [Chevaleyre and Zucker, 2001]. The SVM with the MI kernel achieved 7.0% on "Friendly" and 25% on "Unfriendly" using 20 runs of random leave-10-out [Gärtner et al., 2002]. Note that unlike the Musk datasets, the experimental results between 10-fold CV and leave-10-out are not comparable here because the number of bags is quite different from 100 in both datasets. In Friendly, leave-10-out allows the learner to use more training data whereas in Unfriendly the training data in leave-10-out will be less. In addition, leave-10-out is not stratified, thus the 20 runs used in the SVM with the MI kernel [Gärtner et al., 2002] may suffer from high variance due to the fact that the classes are not evenly distributed in both datasets (see Table 6.1). The two-level classification method (TLC) [Weidmann, 2003], which is a model-oriented metadata-based approach, can achieve 9.31% on Friendly and 18.33% on Unfriendly via 10 runs of 10-fold CV. In Table 6.2, we list the error rates of these methods, together with those of the methods developed in this thesis.

The standard deviation of MILogisticRegressionGEOM of the 10 runs is zero on the "unfriendly" dataset, indicating the stability of the results. The evaluation process allows us to compare the results of the methods developed in this thesis to those of the TLC method [Weidmann, 2003] and of ILP learners [Chevaleyre and Zucker, 2001]. While comparable to the TLC method on the "unfriendly" dataset (actually it seems that MILogisticRegressionGEOM slightly better than TLC on this dataset), MILogisticRegressionGEOM and MI AdaBoost are worse than TLC methods on the “friendly” dataset. However they seem to outperform the ILP learners. The small ridge parameter value $\lambda = 0.05$ used in MILogisticRegressionGEOM on the “friendly” dataset indicates that under the collective assumption, there is indeed a quite strong linear pattern while the larger ridge parameter value used on the “unfriendly” dataset may imply a weaker linear pattern. This is consistent with the observation that the “friendly” data can be fit with linear regression whereas “unfriendly” data cannot. The performances of MILogisticRegressionGEOM and MI AdaBoost are similar on the Mutagenesis datasets, which is reasonable because they are based on the same assumption. As long as the instance-level decision boundary pattern is close to linear under the geometric-average probability assumption, their behaviors will be very similar. The wrapper method with bagged PART is not as good as TLC, MILogisticRegressionGEOM and MI AdaBoost, but still comparable to the best of the ILP methods. Besides, we did not do any parameter tuning or use other improvement techniques like discretization here. Neither did we try the wrapper around other single-instance classifiers. We conjecture that we might be able to improve the accuracy considerably with these efforts because we believe the underlying assumption of the wrapper method (that the class probabilities of all the instances within a bag are very similar to each other) may hold reasonably well in the drug activity prediction problems.

The overall empirical evidence seems to show that the collective assumption may be at least as appropriate in the molecular chemistry domain as the MI assumption. We can get good experimental results on the problems of this kind using methods based on the collective assumption. This empirical observation seems to be contradictory to some claims that the MI assumption is “precisely the case in the domain of molecular chemistry” [Chevaleyre and Zucker, 2000]. However, it would be interesting to discuss this with a domain expert.

	Fruit		
	1	**2**	**3**
Number of positive bags	23	28	19
Number of negative bags	49	44	53
Number of bags		72	
Number of attributes		23	
Number of instances		4560	
Average bag size		63.33	
Median bag size		760	
Minimum bag size		60	
Maximum bag size		120	

Table 6.3: Properties of the Kiwifruit datasets.

6.2 Fruit Disease Prediction

As discussed in Chapter 1, the fruit disease prediction problem provides a MI setting. The identification of this problem as a MI problem was due to Dr. Eibe Frank. In this problem, each bag is a batch of fruits, usually from one orchard. Every fruit within a batch is an instance. Some non-destructive measures are taken for each fruit. The class labels are only observable on the bag level, in other words, either a whole batch of fruits is infected by a disease or not. Although there is no assumption explicitly stated for this problem, the collective assumption is easy to interpret here: if the whole batch of fruits is resistant to a certain disease, then the whole batch is disease-free.

We have obtained one dataset consisting of 72 batches of kiwifruits. they were labeled according to three different diseases, resulting in three datasets with identical attributes, namely Fruit 1, 2 and 3. There are 23 non-destructive measures describing each kiwifruit, resulting in 23 attributes, all numeric. The key properties are listed in Table 6.3. The table shows that number of instances per bag in this dataset is more regular than in the datasets describing the drug activity prediction problem, and comparatively larger. Nonetheless, according to our observation, the number of instances per bag does not seem to be very important in classification. What is more important is the number of bags, which is scarce in this case.

Method	Fruit 1	Fruit 2	Fruit 3
Default accuracy	68.06	61.11	73.61
TLD+EC	62.75±0	52.78 ±0	66.67 ± 0
MILogisticRegressionARITH	71.81±1.47	65.69 ±1.97	73.61 ± 0
MILogisticRegressionGEOM	71.11±2.34	64.03 ±2.31	73.61 ± 0
MI AdaBoost	73.19±1.74	63.61 ±2.84	73.61 ± 0
Wrapper with bagged PART	73.06±0.97	61.81 ±2.87	71.81±0.67

Table 6.4: Accuracy estimates for the Kiwifruit datasets and standard deviations.

One important fact that is not shown in Table 6.3 is that there are some missing values in the dataset. Typically instances' values are missed for a whole bag for a specific attribute. We developed different strategies to deal with the missing values for our methods developed in this thesis. MI AdaBoost has the advantage that the base classifier usually has its own way to deal with missing values, hence it is not a problem for it. For the TLD approach, since we assume attribute independence, we simply skip bags with missing values for a certain attribute when collecting low-order statistics for that attribute. Thus in the log-likelihood function of Equation 5.7 in Chapter 5, we use less bags to estimate the hyper-parameters of that attribute. The cost is that we have less data for the estimation. As for the MI Logistic Regression algorithms, we use the usual way of logistic regression to tackle this difficulty, that is, we substitute the missing values with some values estimated from other training instances without missing values. Again, in order to be consistent to our convention, we substitute the missing values with the weighted average of other instances' values of the concerned attribute. The weight of an instance is given by the inverse of the number of (non-missing) instances of the bag it pertains to.

Now the three datasets can be tackled by all of the methods in this thesis. We do not have other methods to compare (apart from DD, as discussed below), so we list the default accuracy of the data, i.e. the accuracy if we simply predict the majority class, in the first line of Table 6.4 for comparison. However we found out that these datasets are very hard to classify. For example, we tried DD (looking for one target concept) with the noisy-or model and got 10×10-fold CV accuracies of 64.44%(±3.29%) and leave-one-out (LOO) accuracy of 66.67% on the Fruit 1 dataset[1] — worse than the default accuracy. Thus the MI assumption does not seem to hold very well for these datasets. Here we simply present some methods that can give reasonable results on these datasets. We list the accuracy estimates for 10 runs of 10-fold CV in Table 6.4.

As usual, we use the empirical cut point (EC) techniques together with the TLD method. We only show the results of TLD in Table 6.4 because the TLDSimple did worse than TLD in this case. Perhaps in these datasets the mean of each bag is not enough to discriminate the bags from the two classes. Table 6.4 shows that the TLD method, which encodes information about the first two sufficient statistics, cannot achieve the default accuracy either, although the accuracy estimates are extraordinarily stable. It could be the case that in these datasets, higher order sufficient statistics are required for discrimination. It may also be due to the lack of training data as we observed that the LOO accuracy of TLD for the Fruit 1 dataset is 70.83%, which is much better than that shown in Table 6.4 and better than the default accuracy. We have seen in Chapter 5 that the TLD approach is a distributional method and may usually require many bags to provide reasonable estimates. In this case the sample size may be too small to give accurate estimates.

The instance-based methods perform reasonably well on the Fruit 1 and Fruit 2 datasets. It seems that the Fruit 1 dataset has a non-linear instance-level class decision boundary under the collective assumption, while the Fruit 2 dataset exhibits linearity. This is due to the observation that the methods specializing in fitting non-linear functions, like MI AdaBoost and the wrapper method based on bagged PART, can perform better on the Fruit 1 data than the MI linear logistic regression methods, but worse on the Fruit 2 data. Again we did not finely tune the parameters in any of the methods. In the two variations of MI logistic regression, we simply used $\lambda = 1$ for both methods and for all the datasets. In MI AdaBoost, we first tried unpruned C4.5 with the minimal leaf-node instance number of 120 (about twice the average

[1] Since DD does not have a mechanism to deal with missing values, we pre-processed the data, substituting the missing values with the average of the non-missing attribute values. Note that this process may give DD an edge in classification because the resulting training and testing instances are different from the original ones and may provide more information for classification.

bag size). The result for the Fruit 2 dataset presented in Table 6.4 is based on this base classifier running 100 iterations. However in Fruit 1, we observed (again) that one iteration gave a good result for this base classifier, which may indicate that we should use a weaker base classifier. Thus for Fruit 1, we used 250 iterations of decision stumps whose results are shown in Table 6.4. As for the wrapper method, we simply used the default setting in the wrapped single-instance learners, i.e. bagging and PART.

The Fruit 3 dataset seems to be very random and hard to classify. In fact, no method can achieve better than the default accuracy. The MI linear logistic regression methods fail to find a linear instance-level decision boundary under the collective assumption. Any ridge parameter values greater than 1 only resulted in predicting the majority class, which appears to be the best accuracy achievable on this dataset. The same happened for MI AdaBoost. When based on the decision stump, any iteration number greater than 10 resulted in worse test accuracy than the default accuracy, although the training error can be improved with more iterations. Thus the best one can do in this dataset is to predict the majority class.

6.3 Image Categorization

Content-based image categorization involves classifying an image according to the content of the image. For example, if we are looking for images of “mountains”, then any images containing mountains are supposed to be classified as positive and those without mountains as negative. The task is to predict the class label of an unseen image based on its content.

This problem can be represented as an MI problem with some special techniques. The key of such a representation is that we can segment an image into several pixel regions, usually called “blobs”. A bag corresponds to an image, which has the blobs or some combinations of blobs as its instances. Again one bag only has one class label. Now the problem is how to generate features for the instances? The data

	Photo
Number of bags	60
Number of attributes	15
Number of instances	540
Number of positive bags	30
Number of negative bags	30
Average bag size	9
Median bag size	9
Minimum bag size	9
Maximum bag size	9

Table 6.5: Properties of the Photo dataset.

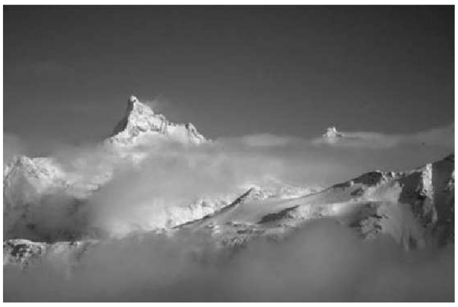


Figure 6.2: A positive photo example for the concept of "mountains and blue sky".

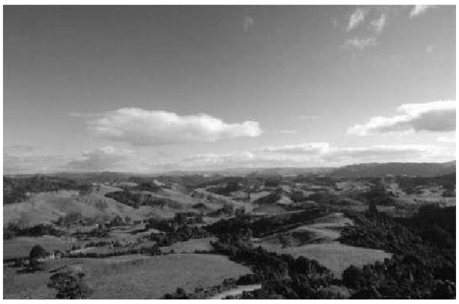


Figure 6.3: A negative photo example for the concept of "mountains and blue sky".

used in this thesis was generated based on the single-blob with neighbours (SBN) approach proposed by [Maron, 1998], although there are many other techniques that can be used [Zhang et al., 2002]. More precisely, there are 15 attributes in the data. The first three attributes are the averaged R, G, B values of one blob. The remaining twelve dimensions are the difference of the R, G, B values between one specific blob and the four neighbours (up, right, down and left) of that blob. As in [Maron, 1998], an image was transformed into 8x8 pixel regions and each 2x2 region was regarded as a blob. There are 9 blobs that can possibly have 4 neighbours in each image. Therefore the resulting data has 15 attributes and 9 instances per bag, fixed for each bag. Details of the construction of the MI datasets from the images can be found in [Weidmann, 2003], where some other methods have also been tried.

Table 6.5 lists some of the properties of the photo dataset we used. Since all the

Method	Photo
Best of the TLC methods	81.67
TLDSimple+EC	69.17±2.64
MILogisticRegressionARITH	71.67±2.48
MILogisticRegressionGEOM	71.00±2.96
MI AdaBoost	76.67±2.36
Wrapper with bagged PART	75±0

Table 6.6: Accuracy estimates for the Photo dataset and standard deviations.

dimensions describe RGB values, they are all numeric. The dataset is about the concept “mountains and blue sky”. There are 30 photos that contain both mountains and blue sky, which are the positive bags, and 30 negative photos. The negative photos could contain only sky, only mountains, or neither. Two examples from [Weidmann, 2003] illustrate the class label properties of the bags. The photo shown in Figure 6.2 is a positive photo, which contains both mountains and the sky, while the one shown in Figure 6.3 is negative because it only contains the sky (and the plains) but no mountains. Note that the classification task here is more difficult than that in [Maron, 1998] and [Zhang et al., 2002] because their target objects only involve one simple concept, say “mountains”, whereas we have a conjunctive concept here, which is more complicated.

Table 6.6 lists the 10×10-fold CV experimental results of some of the methods developed in this thesis. Again we did not finely tune the user parameters. In the MI logistic regression methods, we used a ridge parameter of 2 for both methods. In MI AdaBoost, we used an unpruned C4.5 tree as the base classifier and the minimal instance number for the leaf-nodes is (again) twice the bag size (18 instances). We obtained the results in Table 6.6 using 300 iterations. The base classifiers in the wrapper method were configured using the default settings. We also list the best result for the TLC algorithms [Weidmann, 2003] for comparison.

For this particular dataset, the methods based on the collective assumption may be appropriate sometimes. Because the objects “mountains” and “blue sky” are large objects that usually occupy the whole image, the class label of an image is very likely to be related to the RGB values of all the blobs in that image. However, if this is not the case for an image, the collective assumption may not be suitable. Moreover, the decision boundary in the instance space may be more sophisticated than linear or close to linear (like quadratic) patterns. This is because we take the RGB values as the attributes and the colours of other objects may be very similar to the concept objects (i.e. "mountains" or "blue sky" in this case). To discriminate the positive and negative images, we may need more complex decision boundary patterns. Therefore the TLD approach and the MI logistic regression methods, which model, either asymptotically or exactly, linear or quadratic decision boundaries, are not expected to give good accuracies on this task. MI AdaBoost and the wrapper method (based on appropriate base learners) are more flexible and may be more suitable for this problem.

The natural candidate learning techniques for this problem is the TLC method [Weidmann, 2003], because it can deal with a very general family of concepts, including conjunctive concepts. The DD algorithm (results not shown), on the other hand, cannot deal with this dataset because, although it can be extended to deal with disjunctive concepts, it *cannot* handle conjunctive concepts.

As expected, the best of the TLC algorithms (based on one run of 10-fold CV) indeed performs the best on this dataset. MI AdaBoost and the wrapper method seem to be better than the other methods developed in this thesis because they are able to model more sophisticated instance-level decision boundaries. We also notice that the result of the wrapper method is very stable on this dataset. TLDSimple performs better than TLD and we only present the result for TLDSimple. It produces similar results as the MI linear logistic regression methods. They are not good on this task, as mentioned above, mainly because the instance-level decision boundary is unlikely to be linear (or close to linear), even if the collective assumption holds.

6.4 Conclusion

In this chapter, we have explored some practical applications of MI learning, namely, drug activity prediction, fruit disease prediction and content-based image categorization problems. We experimented with some of the methods developed in this thesis on the datasets related to these problems. We believe that the collective assumption is widely applicable in the real world. This belief is supported by the empirical evidence we observed on the practical problems discussed in this chapter.

Chapter 7

Algorithmic Details

In this chapter we present some algorithmic details that are crucial to some of the methods developed and discussed in this thesis. These techniques relate to either the algorithms or the artificial data generation process. Important features of some algorithms are also discussed.

More specifically, Section 7.1 briefly describes the numeric optimization technique used in some of the methods developed in this thesis. Section 7.2 describes in detail how to generate the artificial datasets used in the previous chapters. Because the instance-based methods we developed in Chapter 4, especially the MI linear logistic regression methods, are quite good at attribute (feature) selection, we show some more detailed results on attribute importance of the Musk datasets in Section 7.3. In Section 7.4 we describe some algorithmic details of the TLD approach developed in Chapter 5. Finally we analyze the Diverse Density [Maron, 1998] algorithm, and describe what we discovered about this method in Section 7.5.

7.1 Numeric Optimization

As already noted above, many methods in this thesis use a numeric technique to solve an optimization problem whenever the solution of the problem is not in a simple analytical form. For example, when the maximum likelihood (ML) method is used (in logistic regression and the TLD approach), we usually resort to numeric optimization procedures to find the MLE. In some situations, we may have bound constraints for the variables, i.e. optimization w.r.t. variables x with constraints $x \geq C$ and/or $x \leq C$ for some constant C. It can be shown that transforming such problems into unconstrained optimization problems via variable transformations like $x = y^2 + C$ or $x = -y^2 + C$ (where y is the new variable) may not be appropriate [Gill, Murray and Wright, 1981]. Because the objective function may not be defined outside the bound constraints, we also need a method that does not require to evaluate the function values there. Eventually we chose a Quasi-Newton optimization procedure with BFGS updates and the "active set method" suggested by [Gill et al., 1981] and [Gill and Murray, 1976]. It is based on the "projected gradient method" [Gill et al., 1981; Chong and Żak, 1996], that is primarily used to solve optimization problems with linear constraints, and that updates the orthogonal projection in each step according to the change of the "active" constraints. In the case of bound constraints, the orthogonal projection of the gradient is very easy to calculate. It is simply the gradients of the "free" variables at that moment (i.e. variables that are not at the bounds). Therefore the searching strategy we adopt is quite similar to that of an unconstrained optimization [Dennis and Schnabel, 1983]. We implemented this optimization procedure in WEKA [Witten and Frank, 1999], as the class `weka.core.Optimization`. The details of the implementation are described in Appendix B. This also serves as a formal documentation of this optimization class.

In order to enhance the efficiency of the optimization procedure and reduce the computational costs, we separate the variables as much as possible. Such a separation of the variables basically divides one optimization problem into several small sub-problems and applies the optimization procedure to each of the sub-problems. Hence in each (smaller) optimization problem, the number of variables to be searched is greatly reduced. Because we use a positive definite matrix to approximate the Hessian matrix in the Quasi-Newton method, reduction of the variables reduces sparsity of the matrix and leads to faster computation. In this thesis, we can sometimes separate the variables to make the optimization easier. For instance, when the parameters of different attributes are separable because we assume the independence of attributes in the likelihood function, we can conveniently divide the likelihood function into sub-functions, usually one for each dimension. Each of these sub-function only involves a few parameters, and we only have to search for the maximum of each sub-function individually, which is a much simpler problem.

7.2 Artificial Data Generation and Analysis

In Chapters 4 and 5, we needed to generate some artificial datasets. To generate these datasets, we needed techniques for generating random variates from some specific distributions. The standard Java library provides us with routines to generate uniform and Gaussian variates but we may need variates generated from other distributions. Specifically, we needed to generate variates in one dimension[1] from a normalized part-triangle distribution in Chapter 4, and from an Inverse-Gamma distribution in Chapter 5. We briefly describe the details here.

Recall that in Chapter 4, we specify the density of X as a triangle distribution and draw data points of one bag from this triangle distribution but within the range of the bag. Since we further normalize the density so that it sums (integrates) to one, we need to generate variates from a normalized part-triangle distribution. Figure 7.1 shows the two cases that can happen for this distribution: (i) is the distribution if the center a bag is outside the interval $[-l/2, l/2]$ where l is the length of the bag range and (ii) is the distribution otherwise. The distribution in case (i) is a line distribution whereas the distribution in case (ii) is a combination of two line distributions. The cornerstone for sampling from both distributions is a line distribution, denoted by $f(.)$. Suppose the line is within $[a, b]$. Then we first draw a point p uniformly in $[a, b)$. If p is in the interval that has greater density, $[a, (a + b)/2)$ in the case shown in Figure 7.1(i), then we accept p. Otherwise we draw another uniform variate v in $[0, f(\frac{a+b}{2}))$. If $v > f(p)$ then we accept $a+b-p$ (i.e. we accept the point symmetric

[1] Because we assumed attribute independence, we generated variates for each dimension and then combined them into a vector.

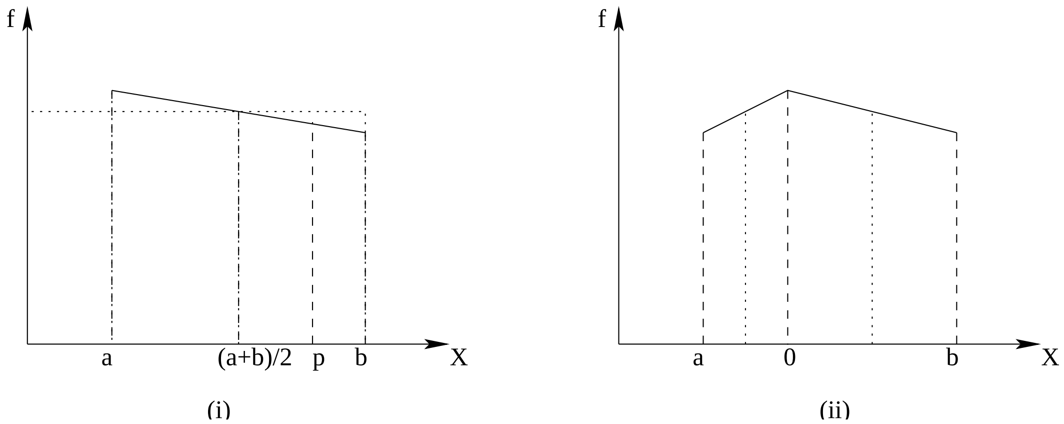


Figure 7.1: Sampling from a normalized part-triangle distribution.

to p w.r.t. $\frac{a+b}{2}$), otherwise we accept p. In such a way we can generate variates with a probability that is equivalent to the area under the line $f(.)$.

When a bag's center is close to 0, we will have the distribution shown in Figure 7.1 (ii). We can view it as a combination of two line distributions — one part in $[a, 0)$ and the other in $[0, b)$. We first decide which part the variate falls into, according to the probabilities that are equivalent to the areas under the two lines.[2] We simply draw a uniform variate within $[0, 1)$ and if it is less than the area under the line in $[a, 0)$, we pick this line distribution, otherwise the other part (in $[0, b)$). For each line distribution we use the same technique as discussed above to draw a random variate from it. This is how we draw data points for each bag from the conditional density given in Equation 4.3 in Chapter 4, where $Pr(x)$ is a triangle distribution.

In the TLD method in Chapter 5, we need to generate σ_k^2 from an Inverse-Gamma distribution (parameterized by a_k and b_k) and μ_k from a Gaussian (parameterized by m_k and $w_k\sigma_k^2$). Then we further generate a bag of instances with a Gaussian parameterized by μ_k and σ_k^2. There is a standard routine in Java for generating a standard Gaussian variate v. Using the transformation $v * \sigma + \mu$ we can get a variate following $N(\mu, \sigma^2)$. For the Inverse-Gamma distribution of σ_k^2, it means that $(\frac{a_k}{\sigma_k^2})$ follows a Chi-squared distribution with b_k degree of freedom (d.f.). Also note that if $X \sim$ *Chi-squared*$(2v)$ with $(2v)$ d.f. , then $Y = \frac{X}{2} \sim$ *Gamma*(v). Therefore we

[2] Note that the total area under the two lines sum to 1.

first generate a variate $u \sim \textit{Gamma}(b_k/2)$ and then by simple transformation we can get σ_k^2 because $\frac{a_k}{\sigma_k^2} = 2u \Rightarrow \sigma_k^2 = \frac{a_k}{2u}$.

Since there is no standard routine in Java to generate standard Gamma variates, we built one from scratch. There are many methods to generate standard Gamma variates [Bratley, Fox and Schrage, 1983; Ahrens and Dieter, 1974; Ahrens, Kohrt and Dieter, 1983]. We chose the one described in [Minh, 1988], and implemented it in `weka.core.RandomVariates` class, which also includes routines generating the standard exponential and Erlang (Gamma with integer parameter) variates.

7.3 Feature Selection in the Musk Problems

First of all, note that we always base our models on certain assumptions, and feature selection is no exception. In this section we discuss feature selection based on the instance-based methods discussed in Chapter 4 under the collective assumptions outlined in Chapter 3. The feature selection is based on the explanatory power of the features for the instance-level class probabilities. In this sense, the "feature selection" discussed here is at the instance level, and is assumption-based. Different assumptions may lead to totally different interpretations of the features.

The Diverse Density [Maron, 1998] (DD) algorithm has also been applied for feature selection on the Musk datasets. It was recognized that the "scaling parameters", one for each attribute, found by DD, indicate the importance of the corresponding attributes — the greater the value of the scaling parameter, the more important the corresponding attribute is. This explanation fits into our understanding of the radial form of the instance-level probability function modeled by DD. The (inverse of) the scaling parameter controls the dispersion of the radial probability function. Intuitively, the larger the dispersion along one dimension, i.e. the smaller the scaling parameter, the "flatter" the class probability around 0.5. Hence this attribute is less useful for discrimination (because a flat probability function implies low purity of the classes on the two sides of the decision boundary). On the contrary, the smaller the dispersion, i.e. the larger the scaling parameter, the "sharper" the probability function along this specific dimension. In other words, this dimension is more useful for discrimination.

However, such a dispersion can be easily scaled. In the probability function for one dimension $\exp[-\frac{(x-p)^2}{s^2}]$, if we multiply both the denominator and numerator by a constant, the probability remains unchanged but we can arbitrarily change the value of the "scaling parameter" s. This means that if we multiplied every data point in one dimension by an arbitrary constant and found the resulting s accordingly, we could effectively manipulate the scaling parameter. Therefore it is necessary to standardize the data before using the scaling parameters as a direct indicator of feature importance. Although [Maron, 1998] divided every data point (along all dimensions) in the Musk datasets by 100 in order to facilitate the optimization, which happened to alleviate the scaling problem, the data points from different dimensions are still on different scales. Thus it may be premature to directly use the scaling parameter values for feature selection.

Formally, in order to test the hypothesis whether one parameter is significant (typically significantly different from 0), we should really find out the sampling distribution of the parameter in question and estimate its standard error from the data. Then we can standardize it to test the significance. However, the (assumed) sampling distribution of the parameter concerned and its standard error are not easy to find, especially in MI datasets. Thus we think it intuitive to use the parameter values found for the standardized data as an indicator. We use this approach to assess the feature importance in the linear logistic regression model for the Musk datasets.

In Figure 7.2 we show the absolute value of the relative linear coefficients found by MILogisticRegressionARITH with a ridge parameter $\lambda = 2$ on the Musk datasets. The coefficients are taken as the relative values to the maximal (absolute value of) coefficient value, thus the largest is 100%. The data was standardized using the weighted mean and standard deviation, as described in Chapter 4. In linear logistic models, the meaning of the coefficients of the attributes is straightforward. If we

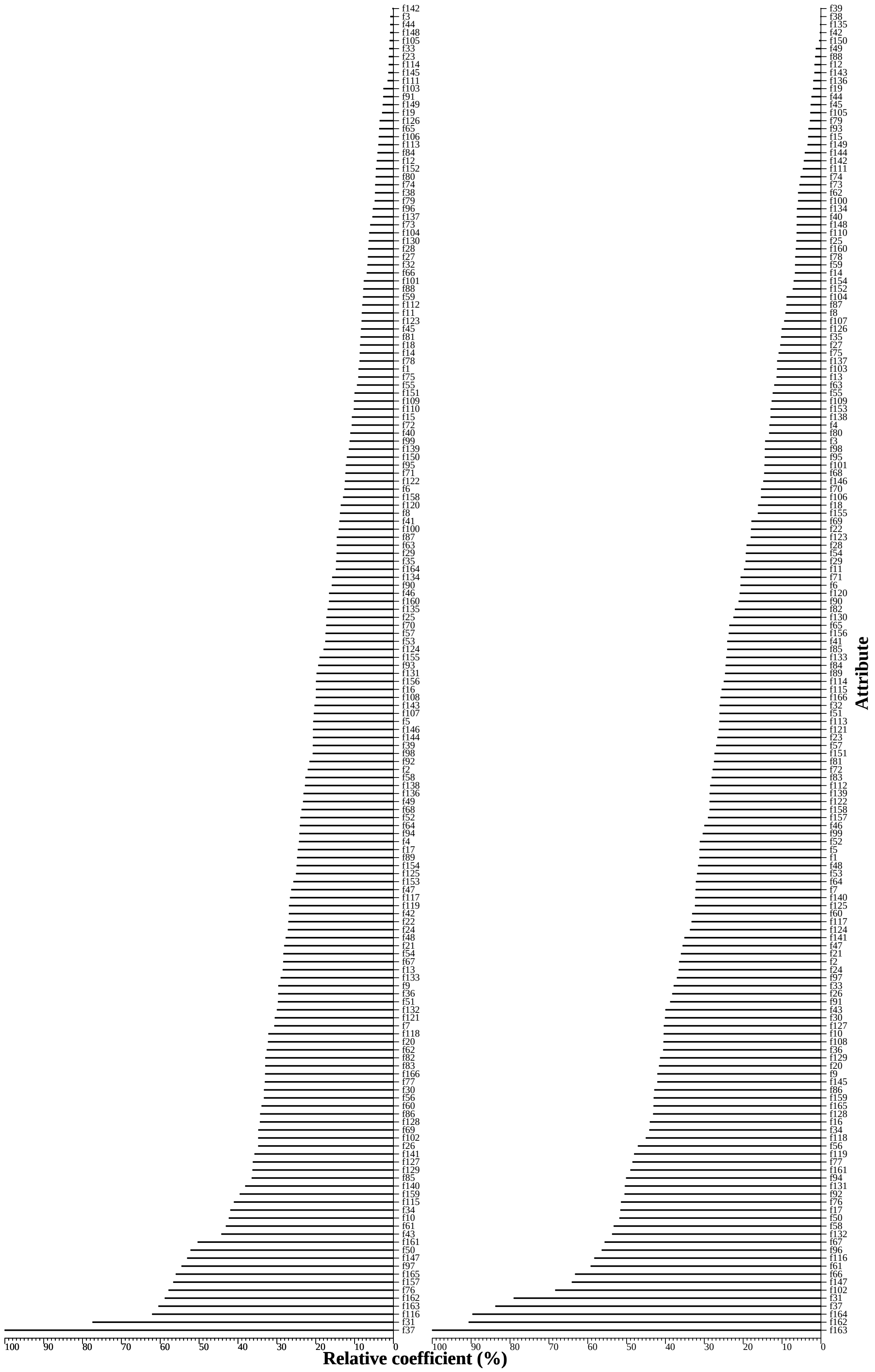


Figure 7.2: Relative feature importance in the Musk datasets: the left plot is for Musk1, and the right one for Musk2.

fix the value of all dimensions but one, and plot the probability function along that one dimension, we will find the familiar shape of the sigmoid (or inverse sigmoid) function, and the absolute value of coefficient controls the sharpness of the function around the value of 0.5. If the absolute value of a coefficient of one attribute is high, then the probability function is very sharp around the value of 0.5. It means that on both sides of the decision boundary, the purity of two classes is high and we can easily separate them with this attribute, and vice versa. Therefore, it is intuitive to regard this as an indicator of the feature importance. Note that in order to avoid the scaling problem mentioned above, we use the coefficients estimated from the standardized data (and do not include the intercept). The ridge method tends to shrink the coefficients to zero, and this is what we observed after the transformation of the coefficients back to original scale. As can be seen in Figure 7.2, the relative coefficients estimated from both datasets do not differ as much between attributes as we might expect from the results presented in [Maron, 1998], although we do observe that some attributes are much more important than others, especially in the Musk1 dataset. If we set a threshold for the relative coefficients to select attributes, say 40% or 50%, we may indeed pick up only a minority of the attributes (based on the collective assumption and the linear logistic instance-level model).

7.4 Algorithmic Details of TLD

There are three important factors in the implementation of the TLD method developed in Chapter 5: the first is the integral in the model; the second is the constrained optimization; and the third is the numeric evaluation of the Gamma function. The second factor has already been addressed in Section 7.1 and we discuss the other two in this section. In addition, we discuss more about the interpretations of the parameters involved in TLD.

For the specific integral calculus for Equations 5.6 and 5.9 in Chapter 5, we list the solution in Appendix C. We then maximize the formula resulting from the integration. However, in general, if we specify arbitrary instance and bag-level distributions, the integration of the instance-level parameters is really hard, if not impossible, which may restrict the practical feasibility of this method. It was thus suggested in the EB literature that an EM algorithm is used for this method [Maritz and Lwin, 1989]. More specifically, using EM terminology, we regard the instance-level parameter θ as the "unobservable data" and the bag-level parameter δ^y as the "observable data" in Equation 5.1 of Chapter 5. Now given a specific value of δ^y_0, we have the probability of θ, $Pr(\theta|\delta^y_0)$ and the "complete data" likelihood function $L(B_j, \theta, \delta^y_0) = Pr(B_j|\theta, \delta^y_0) = Pr(B_j|\theta)$. Thus the integral can be regarded as taking the expectation of the"complete data" likelihood function over the "unobservables", i.e. $E_\theta[L(B_j, \theta, \delta^y_0)]$. This constitutes the E-step in EM and the resulting formula of the expectation is exactly the same as the last line in Equation 5.1. Then we regard the "observables" δ^y as a random variable and maximize this expected likelihood function w.r.t. δ^y, which is the M-step. Under some regularity conditions, we can maximize $L(B_j, \theta, \delta^y)$ within the expectation sign. Hence *within the expectation sign*, we take a Newton step from δ^y_0 to get a new value δ^y_1, that is $\delta^y_1 = \delta^y_0 + E(H^{-1}_L)E(g_L)$ where H^{-1}_L is the Hessian matrix (second derivative) of the likelihood and g_L is the Jacobian (first derivative) at the point δ^y_0. The expectation is, again, over θ and can be numerically evaluated now. This defines an iterative solution of this maximum likelihood method, which was cited by the well-known original paper of the EM algorithm [Dempster et al., 1977] as an early example of the EM algorithm.

In the TLD method we also have a Gamma function to evaluate in Equation 5.6. More precisely, we need to evaluate:

$$g = -log\frac{\Gamma((b_k + n_j)/2)}{\Gamma(b_k/2)}.$$

Define $h = \lfloor \frac{n_j}{2} \rfloor$, and $s = \sum_{x=1}^{h} log(b_k/2 + h - x)$. If n_j is even, $g = -s$ according to the well-known identity $\Gamma(y + 1) = y\Gamma(y)$. But if n_j is odd, we have $g = -s - log\frac{\Gamma(b_k/2+1/2)}{\Gamma(b_k/2)}$, thus we still have log-Gamma function to evaluate. We used the implementation suggested by [Press, Teukolsky, Vetterling and Flannery, 1992] to evaluate $\log\Gamma(y)$. However, since the method we use to search for the

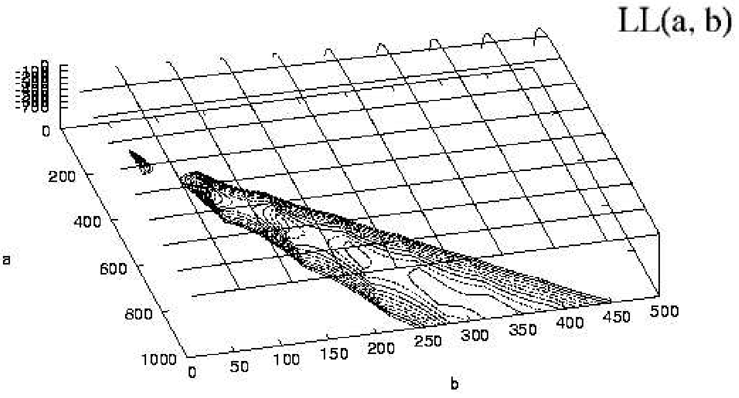


Figure 7.3: Log-Likelihood function expressed via parameter a and b.

minimum of the negative log-likelihood function requires the Jacobian and Hessian matrix of the function,[3] we also need to evaluate $\frac{\mathrm{d}}{\mathrm{d}y}\log\Gamma(y)$ and $\frac{\mathrm{d}^2}{\mathrm{d}y^2}\log\Gamma(y)$ in this case. Fortunately there is an easy approximation for them [Artin, 1964]:

$$\frac{\mathrm{d}}{\mathrm{d}y}\log\Gamma(y) = -C - \frac{1}{y} + \sum_{i=1}^{+\infty}\left(\frac{1}{i} - \frac{1}{y+i}\right)$$

$$\frac{\mathrm{d}^2}{\mathrm{d}y^2}\log\Gamma(y) = \frac{1}{y^2} + \sum_{i=1}^{+\infty}\frac{1}{(y+i)^2}$$

where C is Euler's constant and is canceled out when taking differences as in Equation 5.6.

We would like to further analyze the model in TLD to get more insight into the interpretation of the parameters in this method. We restrict our discussion to one dimension so that we can discard the subscript k. The parameters a and b together define the properties of σ^2. The parameter m, the mean of μ, is quite independent on the other parameters whereas w depends on both the variance of μ and the (expected value of) σ^2. Thus it seems that the most difficult part of the interpretation comes from the parameters a and b. If we fix the values of w and m, and assume some reasonable value for the sample mean $\overline{x}$ and the sample variance $\frac{S^2}{n-1}$, we can get a

[3] Note that the Quasi-Newton method we used itself does not need the Hessian matrix. We provide the Hessian to give better solutions in case of the bound constraints.

log-likelihood (LL) function similar to that in Figure 7.3, which shows the value of the LL function we constructed associated with different values of a and b, denoted by $LL(a, b)$.

Although it looks flat, there *are* maximal points in LL, according to the contour plot on the a-b plane. As a matter of fact, the maximal points seem to lie on a particular line. To explain this, note that σ^2 follows an Inverse-Gamma distribution, alternatively, $(\frac{a}{\sigma^2})$ follows a Chi-squared distribution with b degree of freedom. Thus the density function of σ^2 is:

$$\mathrm{d}F(\sigma^2) = \frac{\left(\frac{a}{2\sigma^2}\right)^{b/2} \exp[-\frac{a}{2\sigma^2}]}{\sigma^2\Gamma(b/2)} \mathrm{d}\sigma^2$$

If we calculate the mean (i.e. the first moment), it is:

$$E(\sigma^2) = \int_0^\infty \frac{\left(\frac{a}{2\sigma^2}\right)^{b/2} \exp[-\frac{a}{2\sigma^2}]}{\Gamma(b/2)} \,\mathrm{d}\sigma^2$$

By setting $y = a/(2\sigma_k^2)$, we have

$$\begin{aligned}
&= \frac{1}{\Gamma(\frac{b}{2})} \int_0^\infty y^{b/2} \exp(-y) \frac{a}{2y^2} \,\mathrm{d}y \\
&= \frac{a}{2\Gamma(\frac{b}{2})} \int_0^\infty y^{\frac{b}{2}-2} \exp(-y) \,\mathrm{d}y
\end{aligned}$$

and if $0 < \frac{b}{2} \neq 1$, we use integration by parts,

$$\begin{aligned}
&= \frac{a}{2\Gamma(\frac{b}{2})} \Big\{ \frac{1}{\frac{b}{2}-1} \big[y^{\frac{b}{2}-1} \exp(-y)|_0^\infty + \int_0^\infty y^{\frac{b}{2}-1} \exp(-y) \,\mathrm{d}y \big] \Big\} \\
&= \frac{a}{2\Gamma(\frac{b}{2})} \Big\{ \frac{1}{\frac{b}{2}-1} \big[y^{\frac{b}{2}-1} \exp(-y)|_{y\to\infty} - y^{\frac{b}{2}-1} \exp(-y)|_{y\to 0} + \Gamma(\frac{b}{2}) \big] \Big\} \\
&= \frac{a}{2\Gamma(\frac{b}{2})} \Big\{ \frac{1}{\frac{b}{2}-1} \big[0 - y^{\frac{b}{2}-1} \exp(-y)|_{y\to 0} + \Gamma(\frac{b}{2}) \big] \Big\}
\end{aligned}$$

If $\frac{b}{2} > 1$, i.e. $b > 2$, $y^{\frac{b}{2}-1} \exp(-y)|_{y\to 0} = 0$, so $E(\sigma^2) = \frac{a}{b-2}$. Otherwise, if $b < 2$,

$y^{\frac{b}{2}-1}\exp(-y)|_{y\to 0} = \infty$ and thus $E(\sigma^2) = \infty$. If $b = 2$,

$$E(\sigma^2) = \frac{a}{2\Gamma(\frac{b}{2})}\int_0^\infty y^{-1}\exp(-y)\,\mathrm{d}y = \infty.$$

Therefore, if $b \leq 2$, the distribution is proper but the mean does not exist, otherwise the mean is $\frac{a}{b-2}$.

Hence as long as the ratio of a and $b-2$ keeps a constant, the (expected) value of σ^2 is the same, and so is the LL function value given other parameters and the data. But note that the variance of σ^2 is not the same even if $a/(b-2)$ is a constant because $Var(\sigma^2) = 2a^2/[(b-2)^2(b-4)]$ provided that $b > 4$. That is why in Figure 7.3 the values of a and b corresponding to the maximal LL values are not exactly linear, but very close to linear.

This analysis is useful for performing feature selection using the TLD method. Even if two features have different a_k and b_k values but similar values of $\frac{a_k}{b_k-2}$, w_k, and m_k, they are still not useful for discrimination. Moreover, the parameter w_k denotes the ratio $\frac{Var(\mu_k)}{\sigma_k^2}$. Given $Var(\mu_k)$, w_k actually depends on a_k and b_k. Experiments (on the artificial data) show that if we specify an independent $Var(\mu_k)$ and $b_k > 2$, the TLD method will find w_k as $Var(\mu_k)/\frac{a_k}{b_k-2}$, which is reasonable according to the above analysis. However, if $b_k \leq 2$, whatever TLD finds will be uninterpretable. Unfortunately, we did find many such attributes in the Musk data, which may be a reason why the simplified model TLDSimple can work better than TLD on this data. On the other hand, when we applied TLD on the kiwifruit datasets described in Chapter 6, we did not observe this (adverse) phenomenon (i.e. all b_k's > 2). As a result, TLD can be applied successfully and, as discussed in Chapter 6, it actually outperformed TLDSimple on this data.

7.5 Algorithmic Analysis of DD

In this section we briefly describe how we can view the Diverse Density (DD) algorithm [Maron, 1998] as a maximum binomial likelihood method, and how we should interpret its parameters.

The Diverse Density (DD) method was proposed by Maron [Maron, 1998; Maron and Lozano-Pérez, 1998] and has EM-DD [Zhang and Goldman, 2002] as its successor, which claims to be an application of the EM algorithm to the most-likely-cause model of the DD method. However, we think that the EM-DD algorithm has problems to find a maximum likelihood estimate (MLE), as shown in Appendix D. Thus we are strongly skeptical about the validity of EM-DD's solution as an ML method. We only discuss the DD algorithm here.

The basic idea of DD is to find a point x in the instance space such that as many positive bags as possible overlap on the point but no (or few) negative bags cover it. The "diverse density" of a point x stands for the probability for this point to be the "true concept". Mathematically,

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

where B is one bag. The "+/−" sign indicates a bag's class label and t is the true concept. DD aims to "maximize this probability with respect to t, to find the target concept c_t that is most likely to agree with the data"[Maron, 1998]. After some manipulations, it is equivalent to maximizing the following "likelihood" function

$$L(x|\mathbf{B}) = \prod_{1 \leq i \leq n} Pr(x = t | B_i^+) \prod_{1 \leq j \leq m} Pr(x = t | B_j^-) \qquad (7.1)$$

We believe this notation is confusing and may even disguise the essence of the DD method. Hence we would like to express this method using a different perspective. As is usually done in single-instance learning, we would like to explicitly represent the class labels as a variable Y. In MI, there is still a Y, not for each instance, but for each bag. In general $Y = 1, 2, \cdots, K$ for K-class data but the Musk datasets are two-class data. Here we say $Y = 0$ if the bag is negative and 1 if it is positive. In the above formulation this variable is disguised by the "$+$" and "$-$" sign and we believe that $Pr(x = t|B_i^+)$ really means $Pr(Y = 1|B_i)$, and likewise $Pr(x = t|B_j^-)$ means $Pr(Y = 0|B_j)$.

[Maron, 1998] proposes two approximate generative models, namely the noisy-or and the most-likely-cause models, to calculate $Pr(x = t|B_i^+)$ and $Pr(x = t|B_i^-)$ in practical problems. They can be described as follows:

- **Noisy-or model:** DD borrows the noisy-or model from the Bayesian networks' literature. Noisy-or calculates the probability of an event to happen in the face of a few causes, assuming that the probability of any cause failing to trigger this event is independent of any other causes. DD regards an instance as a cause and "c_t being the underlying concept" as the event, hence using the original notation

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

  and

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

- **Most-likely-cause model:** In the most-likely-cause model, only one instance in each bag is regarded as the cause for making a certain x the "true concept", say, the z_i^{th} instance in the i^{th} bag. z_i is defined as

  $$z_i = argmax_{z_i}\{Pr(x = t|B_{iz_i})\}.$$

  As before, we have the two complementary expressions as

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

and

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

A well-known category of single-instance learners model the posterior probability of Y given the feature data X, $Pr(Y = k|X)$ ($k = 1, 2$) directly, e.g. logistic regression or entropy-based decision trees. Logistic regression models the process that determines an instance's class label as a one-stage Bernoulli process and hence fits a one-stage binomial distribution for $Pr(Y = k|X)$. It then uses the ML method to estimate the parameters involved. According to our understanding, DD still uses the above single-instance paradigm. Each point (i.e. each instance) in the instance space has a probability of being positive and negative and DD models the posterior probability of Y directly. The difference is that we now have multiple instances associated with one class label.

In the noisy-or model, we model the process that determines the class label of a bag as an m-stage Bernoulli process where m is the number of instances inside the bag — each one of the instances corresponds to one stage. Suppose we knew the probability for each instance to be positive $Pr(Y = 1|x_b)$ ($b = 1, 2, \cdots, m$), then we could first determine the class label of each instance (or stage) according to its class probability. Given this, what is the probability for *all* the instances (stages) to be negative if they are *independent*? This is a simple question and the solution is $\prod_{b=1}^m (1 - Pr(Y = 1|x_b))$. The complementary probability $1 - \prod_{b=1}^m (1 - Pr(Y = 1|x_b))$ corresponds to the probability for *at least one of them* to be positive. This is a probabilistic representation of the standard MI assumption. Now the log-likelihood function of this multi-stage Binomial probability is simply

$$L = \sum_B [Y \log(1 - \prod_{b=1}^m (1 - Pr(Y = 1|x_b))) + (1 - Y) \log(\prod_{b=1}^m (1 - Pr(Y = 1|x_b)))] \tag{7.2}$$

where there are B bags in total. This is exactly what DD with the noisy-or model uses. Within this multi-stage Bernoulli process, it is straightforward to write down other formulae if the conditions for the MI assumption change, for instance, as "a bag is positive if at least r ($1 < r < m$) instances are positive and negative otherwise", etc.

The most-likely-cause, on the other hand, selects a representative of each bag based on $Pr(Y = 1|x_b)$. More specifically, $b = argmax_{b \in m}\{Pr(Y = 1|x_b)\}$, i.e. it selects the instance with the highest probability to be positive in a bag. Therefore it literally degrades a bag into one instance and the one-stage Bernoulli process is applied to determine the class label, as in the mono-instance case. The log-likelihood function is now

$$\begin{aligned} L = \sum_B [Y \log(max_{b \in m}\{Pr(Y = 1|x_b)\}) + \\ (1 - Y) \log(1 - max_{b \in m}\{Pr(Y = 1|x_b)\})] \\ = \sum_B [Y \log(max_{b \in m} Pr(Y = 1|x_b)) + (1 - Y) \log(min_{b \in m} Pr(Y = 0|x_b))] \end{aligned} \tag{7.3}$$

where there are B bags in total. This is exactly what DD with the most-likely-cause model uses. The most-likely-cause also follows the standard MI assumption because, by selecting an instance in a negative bag that has the maximal probability to be positive and setting that probability (via the binomial model) to less than 0.5, it implies that the probability to be positive for every instance in a negative bag cannot be greater than 0.5.

This means that DD uses the binomial probability formulation, either one stage or multiple stage, to model the class probabilities of the bags. In general, we can separate the modeling into two levels if we introduce a new variable denoting the bags B. On the bag level we always have a one-stage binomial formulation for $Pr(Y|B)$ and on the instance level we build a relationship between $Pr(Y|B)$ and $Pr(Y_X|X)$ where Y_X is an instance's class label, which is unobservable from the data. Mathematically, at the bag level, assuming i.i.d data,[4] the marginal probability

[4] We assume that the class labels Y_{B_i} of the bags B_i are independent and identically distributed (i.i.d) according to a binomial (or multinomial in a multi-class problem) distribution.

of Y is a one-stage binomial distribution,

$$Pr(Y|B_i) = \theta^{Y_{B_i}}(1-\theta)^{1-Y_{B_i}}$$

where $\theta = Pr(Y = 1|B_i)$. Normally we have a parametric model for θ with a parameter vector β estimated using the data, i.e. $\theta = g(\beta, B_i)$ in this case. Thus we can regard the marginal probability of Y as being parameterized by β. In other words, the likelihood function for β is

$$\begin{aligned}\mathbf{L}(\beta|\mathbf{Y_B}) &= Pr(Y_{B_1}, \cdots, Y_{B_N}|B_1, \cdots, B_N, \beta) \\ &= \prod_{1\le k\le N} g(\beta, B_k)^{Y_{B_k}}(1-g(\beta, B_k))^{1-Y_{B_k}},\end{aligned}$$

and, assuming n positive bags and m negative bags,

$$= \prod_{1\le i\le n} g(\beta, B_i) \prod_{1\le j\le m} (1-g(\beta, B_j)). \tag{7.4}$$

At the instance level, we build a relationship between $Pr(Y|B_k)$ and $Pr(Y_X|X_{kl})$ with $X_{kl} \in B_k$, i.e. $Pr(Y|B_k) = h(Pr(Y_X|X_{kl})) \Rightarrow g(\beta, B_i) = h(f(\beta, X_{kl}))$ where $f(\beta, X_{kl}) = Pr(Y_X|X_{kl})$. In the noisy-or model, $h(f) = 1 - \prod_{l=1}^{n_k}(1 - f(\beta, X_{kl})$. In the most-likely-cause model, $h(f) = max_l\{f(\beta, X_{kl})\}$. One could plug in other $h(.)$'s based on other assumptions believed to be true but the binomial likelihood function in Equation 7.4 remains unchanged. This perspective on DD establishes its relationship with the MI methods described in Chapter 4.

The likelihood in Equation 7.4 is a generalization of Equations 7.2 and 7.3, and it is identical to the “likelihood” function in Equation 7.1 that was given in the original description of DD. Now if we think of β as fixed and estimate $\hat{\beta}$ by maximizing the likelihood function, it is easily recognized that DD is simply a maximum binomial likelihood method.

The last question to ask is how to establish an exact formula for the instance-level probability $Pr(Y_X|X_{kl}) = f(\beta, X_{kl})$. [Maron, 1998] proposed three ways. One is to use $\exp(-||X_{kl} - p||^2)$ where $||.||$ is the Euclidean norm and p is the parameter standing for a point in the instance space. The second one is to use $\exp(-||s(X_{kl} - p)||^2)$ where s is a diagonal matrix with diagonal elements that are the scaling parameters for the different dimensions. The last model is a variation of the second one that models a set of disjunctive concepts, each of which is the same as the second model. As a matter of fact, suppose we knew there are D concepts to be found. Then the DD method would have D sets of parameters (instead of one set of parameters) $v_1, v_2, \cdots, v_D$, where v_d, $d = 1, 2, \cdots, D$, is a vector of parameters consisting of both point and scaling parameters (p and s) for each dimension. In the process of searching for the values of the p_d's and s_d's, the probability of each X_{kl} is associated with only one concept — the one that makes X_{kl} to have the highest $Pr(Y = 1|X_{kl})$. In other words, $Pr(Y = 1|X_{kl})$ is calculated as $max_d\{\exp(-||s_d(X_{kl} - p_d)||^2\}$.

The formulation of $Pr(Y_X|X_{kl})$ ($f(\beta, X_{kl})$) is in a radial (i.e. "Gaussian-like") form, with a center of p_t and a dispersion of $\frac{1}{s_t}$ in the t^{th} dimension (where s_t is the t^{th} diagonal element in s). The closer an instance is to p, the higher its probability to be positive. And the dispersion determines the decision boundary of the classification, i.e. the threshold of when $Pr(Y_X|X_{kl}) = 0.5$. It is similar to the axis-parallel hyper-rectangle (APR) [Dietterich et al., 1997] on the instance level. But APR is not differentiable. In order to make it differentiable, DD essentially models the (instance-level) decision boundary as an axis-parallel hyper-ellipse (APE) instead of a hyper-rectangle. The diameter of this APE along the t^{th} dimension is $\frac{\sqrt{\log 2}}{s_t}$. For example, in a two-dimensional space, the decision boundary is

$$\exp[-s_1^2(x_1 - p_1)^2 - s_2^2(x_2 - p_2)^2] = 0.5 \Rightarrow \frac{(x_1 - p_1)^2}{\frac{\log 2}{s_1^2}} + \frac{(x_2 - p_2)^2}{\frac{\log 2}{s_2^2}} = 1$$

where p_1, p_2, s_1, s_2 are the parameters to be estimated. We know this is an ellipse centered at p_1 and p_2, and being $\frac{\log 2}{s_1^2}$ and $\frac{\log 2}{s_2^2}$ in diameter along the two axes. Any point within this ellipse should be classified as positive. This is exactly the second formulation in DD described above. The third model in DD models more than one

APE using $f(\beta, X_{kl})$. No matter what the formulation for $f(\beta, X_{kl})$ is, note that β is an instance-level parameter and DD aims to “recover” the instance-level probability in a structured form under the MI assumption. Hence we categorize the DD method as an instance-based approach.

Nonetheless, DD interprets the parameter vector s purely as a scaling parameter and does not recognize it as related to the diameter of the decision boundary. As a result it never uses it for classifying a new exemplar (bag). Instead it tries to find new axis-parallel thresholds via an additional time-consuming optimization procedure. We regard this as unnecessary because the instance-level probability has already been recovered (parameterized by p and s) and why not use it? We hence suggest that all the parameters are simply plugged into the noisy-or (or most-likely-cause) model to calculate the binomial probability of $Y_{B_{new}}$ for a new bag B_{new}. The classification is made depending on whether this probability is greater than 0.5. We have done some experiments with DD based on the noisy-or model but without searching for the threshold (i.e. using 0.5 as the threshold) and found that the 10 runs of 10-fold cross validation (CV) accuracy of the DD method is 87.07%±1.40% on the Musk1 data and 83.24%± 2.29% on the Musk2 data. These are very similar to the best results reported when searching for the threshold, which are 88.9% on Musk1 data and 82.5% on Musk2 data,[5] but the computational expense is greatly reduced. The misunderstanding of the parameter s may have also compromised the feature selection in DD, which was discussed in Section 7.3.

Finally we discuss the optimization problem in the ML method in DD. There are no difficulties to numerically maximize the “likelihood” function with the noisy-or model. L in Equation 7.2 can be maximized directly via a numeric optimization procedure. But in the most-likely-cause model, there are max functions in the likelihood of Equation 7.3, which makes it not differentiable. The “softmax” function is used in DD, which is the standard way to make the max function differentiable. The EM-DD method [Zhang and Goldman, 2002] was proposed to overcome the

[5] Because [Maron, 1998] did not report how many runs of 10-fold CV were used, and neither the standard deviation of the accuracies, we cannot do a significant test to see whether the differences are significant.

difficulty of non-differentiability and to make this model faster. However, as shown in Appendix D, it has problems to find the MLE.

Even with the noisy-or model, DD still has a difficult global optimization to solve due to the radial form of $Pr(Y_X|X_{kl})$. The usual way to tackle the global optimization problem is to try different initial values when searching for the optimal value of the variables. [Maron, 1998] proposed a strategy to start searching with the value of every instance within all the positive bags. However this strategy is computationally too expensive to be practical, especially on large datasets like Musk2. [Maron, 1998] also mentioned that, theoretically, to start from every instance within one positive bag can be enough to approximately find the point with the highest diverse density. We thus adopt the latter strategy in our implementation of DD (in the `MI` package, as described in Appendix A). More precisely, we picked up the positive bag(s) with the largest size (we picked all of them if there are more than one), and tried every instance within the bag(s) as the start value of the search. In fact we observed that this strategy gives a higher accuracy than the strategy that starts with instances' values from all the positive bags on the Musk1 data. However, the improvement proposed in Section 4.6 of Chapter 4, namely to change the formulation of $Pr(Y_X|X_{kl})$, can help avoid this inconvenience in the optimization process.

In summary, we recognize DD as a parametric method that uses the maximum binomial likelihood method to recover the instance-level probability function in an APE form based on the MI assumption. Therefore it is a member of the "APR-like + MI assumption" family.

Chapter 8

Conclusions and Future Work

The approach adopted in this thesis is a "conservative" one in the sense that it is similar to existing methods of multiple instance learning. Much of the work is an extension or a result derived from the statistical interpretation of current methods in either single-instance or MI learning. Although some new MI methods have been described in this thesis, we basically adopted a theoretical perspective similar to that of the current methods. Hence much of the richness of the multiple instance problems is left to be explored. In this chapter, we first summarize what has been done in this thesis. Then we propose what could be done to more systematically explore MI learning in a statistical classification context.

8.1 Conclusions

In this thesis, we first presented a framework for MI learning based on which we summarized MI methods published in the literature (Chapter 2). We defined two main categories of MI methods: instance-based and metadata-based approaches. While instance-based methods focus on modeling the class probability of each instance and then combine the instance-level probabilities into bag-level ones, metadata-based methods extract metadata from each bag and model the metadata directly. Note that in the instance-based methods, the combination of the instance-level predictions into bag-level ones requires some assumptions.

The standard assumption that can be found in the literature is the MI assumption. We proposed a new assumption instead of the MI assumption and called it the "collective assumption". We also explained that some of the current MI methods have implicitly used this assumption. Under the collective assumption, we developed new methods that fall into two categories: bag-conditional and group-conditional approaches.

A bag-conditional approach models the probability of a class given a bag of n instances $Pr(Y|X_1, \cdots, X_n)$ (or some transformation of the probability). Under the collective assumption we can model it as some function $f[.]$ of the point-conditional probability $Pr(Y|X)$ (or a transformation of this probability), i.e.

$$Pr(Y|X_1, \cdots, X_n) = f[Pr(Y|X_i)], i = 1, \cdots, n.$$

Because many single-instance learners model $Pr(Y|X)$ (or a transformation of it), we can either wrap around them (Chapter 3) or upgrade them (Chapter 4) to enable them to deal with MI data. The resulting methods are instance-based MI learners.

A group-conditional approach models the probability density of a bag of n instances given a class, i.e. $Pr(X_1, \cdots, X_n|Y)$, and then calculates $Pr(Y|X_1, \cdots, X_n)$ based on $Pr(X_1, \cdots, X_n|Y)$ and Bayes' rule. It is not obvious how to model the density $Pr(X_1, \cdots, X_n|Y)$ directly. Under the collective assumption, we could have simply assumed $X_1, \cdots, X_n$ are from the same density. However, the generative model would have been too simple to solve real-world problems. Instead, we assumed that all instances from the same bag are from the same density while different bags correspond to different (instance-level) densities. We then related the parameters of these densities to one another by using a hyper-distribution (or bag-level distribution) on the parameters. This resulted in a two-level distribution (TLD) solution (Chapter 5) to MI learning. This is essentially a metadata-based approach. We discovered that this approach is an application of the empirical Bayes method from statistics to the MI classification problem.

Then we explored some practical applications of MI learning — the drug activity prediction, fruit disease prediction and content-based image categorization (Chapter 6). We also performed some experiments on datasets from these practical domains and found that the methods developed in this thesis are competitive with published methods.

Finally we presented some important algorithmic details in the methods discussed in this thesis (Chapter 7). These include numeric optimization techniques, artificial data generation details, feature selection on the Musk datasets, algorithmic details of the TLD method, and the analysis of the DD algorithm [Maron, 1998].

As a by-product of this thesis, we also discovered the relationship between MI learning and the meta-analysis problem in statistics (Section 5.6 in Chapter 5), notified some errors in some of the current MI methods (Appendix D) and implemented some numeric procedures for the WEKA workbench [Witten and Frank, 1999] (Chapter 7 and Appendix B).

8.2 Future Work

MI learning differs from single-instance learning in two ways: (1) it has multiple instances in an example, and (2) only one class label is observable in the data for each bag of instances. Although the name "multiple instance" seems to denote only the first property, it has become a convention in MI learning that both should be satisfied. Let us factorize these two ways into two steps, which may help us see a direction for future work on MI learning with a statistical perspective.

Learning problems with multiple instances per bag

First, let us consider a problem simpler than the MI problem—we have multiple instances in an example, but each instance has its own class label. In other words, we construct the data as in single-instance learning, adding one more attribute named

"Bag ID" that indicates which bag an instance is from. At testing time, a new bag of instances is given but each instance is to be classified individually. The reader might think that this is an uninteresting problem because we could apply single-instance learners directly to solve this problem by deleting the "Bag ID". However, this line of thinking may not be true. If the fact that some instances are from the same bag indeed provides us with some additional information about their class labels, *none* of the single-instance learners can perform well on this problem because they all ignore this extra information that implicitly resides in the data.

For example, suppose the posterior (class) probability of each instance is dominated by some parameter β, $Pr(Y|X, \beta)$, where X includes all the attributes *except* the "Bag ID" attribute. Now suppose β changes from bag to bag, following a specific distribution. Then we have β_1, for each instance in the first bag and can generate its instances' class labels according to $Pr(Y|X, \beta_1)$, β_2 for another bag, generating its instances' class labels based on $Pr(Y|X, \beta_2)$, etc. Obviously normal single-instance learners are not expected to deal with this data because they cannot use the information provided by the "Bag ID" attribute. Since this information resides in the data, there is room to develop a new family of methods that can fully utilize the bag information. Such new methods may outperform normal single-instance learners on this type of problems.

We call such a problem a "semi-MI" problem because the second property of MI problems is not satisfied. As shown above, much of the richness of multiple instances learning already appears in semi-MI problems where normal single-instance learning cannot be applied. When classifying a test instance in semi-MI learning, we can regard the rest of the instances within the same bag as an "environment" for the classification. Even if the instance to be classified does not change, the classification may change if the "environment" (i.e. other instances within the bag) changes. Ignoring this contextual information may not give an accurate prediction.

Nevertheless MI research seems to regard the semi-MI problem as the same setting as normal single-instance learning, and semi-MI problems do not appear to be actively researched. Almost all of the current MI methods and the methods in this thesis (except the TLD approach) have *not* thoroughly and systematically explored the extra information provided purely from the setting of multiple instances per example. Therefore we regard this as the first step to tackle MI problems in the future.

Note that even when there is only one instance per bag, i.e. the data degenerates into mono-instance data, methods that treat instances as degenerated bags may be totally different from normal single-instance learners. There is some work in normal single-instance learning that already has such a perspective. Such methods can be shown to have some asymptotic optimal properties [Wang and Witten, 2002].

The setting of multiple instances per bag is not restricted to the classification case. It can be extended naturally to regression and clustering, which may be more commonly seen in practical problems. Therefore we strongly advocate the study of "semi-MI" problems in the MI domain.

One class label for a bag

Once we have fully explored the richness of semi-MI problems, we can consider MI problems where the instances' labels are not observable. This can be, for example, based on some assumptions that relate a bag's class label to the corresponding instances' class labels. The MI assumption has been adopted by many current (instance-based) MI methods, and the collective assumption is adopted in this thesis. Future work is likely to focus on these assumptions made. The study of assumptions can follow two directions: (1) the creation and formulation of new assumptions, and (2) the interpretation and assessment of existing assumptions.

Note that the categorization in our framework described in Chapter 2 is actually highly related to the assumptions. In instance-based methods, the underlying assumptions are purely related to the (unobservable) instances' class labels, while in metadata-based methods the assumptions are, partly or solely, associated with the attribute values of the instances. Note that if the assumptions are no longer associated with the instances' (latent) class labels (as in metadata-based methods' generative models), the problem is *not* related to either single-instance or semi-MI learning because, whether the instances' class labels exist or not, the bags' class labels are generated by some procedures *irrelevant* to the instances' labels. In the future, more assumptions can be created within this framework. Usually the domain knowledge gives rise to these assumptions, and the assumptions reside *outside* the data. The prediction could benefit from incorporating some forms of background knowledge. A common way to incorporate background knowledge is to formulate it mathematically in the model, thus we are typically interested in the exact formulation of the assumptions involved.

The second avenue of future work regarding assumptions can be to assess and interpret existing assumptions, using both domain knowledge and data. Currently the assessment of the validity of the assumptions on a specific dataset is performed via prediction accuracy on the data. However, there is a dilemma sometimes. On the one hand, methods based on seemingly sound domain knowledge may not perform well on corresponding datasets. On the other hand, methods that perform well on practical datasets may be based on some assumptions whose interpretation in the corresponding domain is not straightforward. Therefore we need to acquire both strong background knowledge and modeling skills to fully understand some assumptions. Such efforts may lead to breakthroughs in the understanding of the domain and in the understanding of the learning algorithms.

Applications

We expect that multiple instance learning will keep attracting researchers, mainly due to its prospective applications in various areas. However, one of the biggest obstacles is the lack of fielded applications and publicly available datasets. More MI datasets and practical applications would stimulate research in real world problems for MI learning. In fact, we observed that there are many datasets in which instances are grouped into “bags” while each instance has its own class label. Hence “semi-MI” learning may actually be more promising in terms of applications in the real world.

On the whole, we regard research to MI learning as still in its early stages. Much work on algorithm development, property analysis and practical applications remains to be done.

Appendix A

Java Classes for MI Learning

We have developed a multiple instance package using the Java programming language for this project. A schematic description is shown in Figure A.1. This package is directly derived or modified from the corresponding codes in the WEKA workbench [Witten and Frank, 1999].[1] We put the programs for the MI experiment environment and MI methods directly into the `MI` package, some artificial data generation procedures into the `MI.data` sub-package, and some data visualization tools into the `MI.visualize` sub-package. Although the documentations of the programs are self-explanatory, we briefly describe some of them in the following.

A.1 The "MI" Package

We first developed an experiment environment for MI learning, which includes evaluation tools, some interfaces and related classes. Some of the classes are:

- `MI.MIEvaluation` : A bag-level evaluation environment for MI algorithms.
- `MI.Exemplar` : The class for storing an exemplar, or a bag. Each exemplar has multiple instances but only one class label.

[1] As a matter of fact, we copied some of the source codes from the WEKA files.

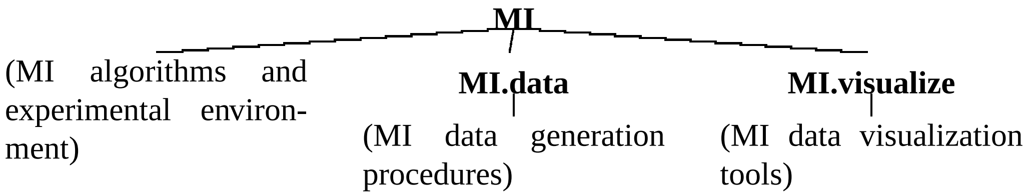

MI
(MI algorithms and experimental environment)
MI.data
(MI data generation procedures)
MI.visualize
(MI data visualization tools)

Figure A.1: A schematic description of the MI package.

- `MI.Exemplars` : The class holding a set of exemplars.

- `MI.MIClassifier` : An interface for any MI classifier that can provide a prediction given a test exemplar.

- `MI.MIDistributionClassifier` : An interface for any MI classifier that can provide a class distribution given a test exemplar. It extends `MI.MIClassifier`.

All the MI methods developed in this thesis as well as some other MI methods are implemented in the `MI` package. More precisely, they are:

- `MI.MIWrapper` : The wrapper method described in Chapter 3.

- `MI.MILRGEOM` : MILogisticRegressionGEOM described in Chapter 4.

- `MI.MILRARITH` : MILogisticRegressionARITH described in Chapter 4.

- `MI.MIBoost` : MI AdaBoost described in Chapter 4.

- `MI.TLD` : TLD described in Chapter 5.

- `MI.TLDSimple` : TLDSimple described in Chapter 5.

- `MI.DD` : The Diverse Density algorithm with the noisy-or model [Maron, 1998] that looks for one target concept. Details are described in Chapter 7.

A.2 The "MI.data" Package

The sub-package `MI.data` includes the classes that generate the artificial datasets used in this thesis:

- `MI.data.MIDataPopulation` : This class uses the population version of the generative models described in Chapter 3 (not used in this thesis).

- `MI.data.MIDataSample` : This class implements the sample version of the generative models described in Chapter 3. The artificial data generated by this procedure is used in Chapters 3 and 4, with different formulations of the density of the feature variable X, $Pr(X)$.

- `MI.data.TLDData` : This class generates the data according to what the method "TLD" models. In particular, given the parameters provided by the user, it first generates the variance of a bag σ^2 according to an Inverse-Gamma distribution parameterized by the user parameters and the mean of that bag μ based on a Gaussian distribution parameterized with the user parameters m, w and the generated variance σ^2 (i.e. generate μ from $N(m, w\sigma^2)$). Finally, it generates a bag of instances according to a Gaussian parameterized by μ and σ^2. The datasets generated by this class are used to show the estimation properties of TLD in Chapter 5.

- `MI.data.TLDSimpleData` : This class generates the data that fits what the method "TLDSimple" models. The details of the data generation process and the generated data are shown in Chapter 5. The generated datasets are used to show the estimation properties of TLDSimple.

- `MI.data.BagStats` : This is a class written by Nils Weidmann, with some functionality added by myself, to summarize bag information for a dataset. It was used for the descriptions of the datasets in Chapter 6.

A.3 The "MI.visualize" Package

In the sub-package of `MI.visualize`, we developed some visualization tools to visualize MI datasets. The key class is `MI.visualize.MIExplorer`, in which we implemented a data plot and a distribution plot for MI datasets. The data plot is used to provide a 2D visualization of a dataset, with the "Bag ID" of each instance clearly specified. Some modifications of this plot were used in Chapters 3, 4 and 5 to illustrate the artificial datasets. The distribution plot is trying to capture the distributional properties within each bag, if possible. Thus it draws a distribution per bag using a kernel density approach. This type of plot was not used in this thesis.

Appendix B

weka.core.Optimization

This appendix serves as a documentation of the `weka.core.Optimization` class. Interested readers or users of this class may find the description here helpful to understand the algorithm.

In brief, the strategy we used is a "Quasi-Newton method based on projected gradients", which is primarily used to solve optimization problems subject to linear inequality constraints. The details of the method are described in the following.

First of all, let us introduce the Newton method to solve an unconstrained optimization problem. We shall convince ourselves, without a proof, that the following procedures can find at least a local minimum if the objective function is smooth.[1] The rigorous proof can be found in various optimization books like [Chong and Żak, 1996], [Gill et al., 1981], [Dennis and Schnabel, 1983], etc.

1. Initialization. Set iteration counter k=0, get initial variable values $\mathbf{x_0}$, calculate the Jacobian (gradient) vector $\mathbf{g_0}$ and the Hessian (second derivative) matrix $\mathbf{H_0}$ using $\mathbf{x_0}$.

2. Check $\mathbf{g_k}$. If it converges to 0, then STOP.

3. Solve for the search direction $\mathbf{d_k}$, $\mathbf{H_k d_k} = -\mathbf{g_k}$. Alternatively this step can be expressed as $\mathbf{d_k} = -\mathbf{H_k}^{-1}\mathbf{g_k}$

[1] One can easily get the update in Step 4 below by Taylor series expansion to the second order from $\mathbf{x_k}$.

4. Take a step to get new variable values $\mathbf{x_{k+1}}$. Normally, this is done as one Newton step $\mathbf{d_k}$, however, when the variable values are far from the function minimum, one Newton step may not guarantee a decrease of the objective function even if $\mathbf{d_k}$ is a descent direction. Thus we are looking for a multiplier α such that $\alpha = argmin(f(\mathbf{x_k} + \alpha\mathbf{d_k}))$ where $f(.)$ is the objective function to be minimized. The search for α is carried out using a line search and once it is done, $\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha\mathbf{d_k}$.

5. Calculate the new gradient vector $\mathbf{g_{k+1}}$ and the new Hessian matrix $\mathbf{H_{k+1}}$ using $\mathbf{x_{k+1}}$.

6. Set k=k+1. Go to 2.

As a convention, $\mathbf{d_k}$ is often referred to as "newton direction" or simply "direction", α as "step length", and $\Delta\mathbf{x} = \alpha\mathbf{d_k}$ as "newton step" or simply "step". We adopt these terms here.

Note that in Step 4 above, we search for the exact value of α that minimizes the objective function. Such a line search is called an "exact line search", which is computationally expensive. It was recommended that only a value of α that can lead to a sufficient decrease in the objective function is needed. In other words, an "inexact line search" is preferable in practice and more computational resources should be put into searching for the values of $\mathbf{x}$ instead of α [Dennis and Schnabel, 1983; Press et al., 1992]. Thus we use an inexact line search with backtracking and polynomial interpolations [Dennis and Schnabel, 1983; Press et al., 1992] in our project.

Now we carry on with Quasi-Newton methods. The idea of Quasi-Newton methods is *not* to use the Hessian matrix because it is expensive to evaluate and sometimes it is not even available. Instead a symmetric positive definite matrix, say $\mathbf{B}$, is used to approximate the Hessian (or the inverse of the Hessian). As a matter of fact, it can be shown no matter what matrix is used, as long as it is symmetric positive definite, and an appropriate update of the matrix is taken in each iteration, the search result will be the same! [Gill et al., 1981] Thus the key issue is how to update this matrix $\mathbf{B}$. One of the most famous methods is a variable metric method called “Broyden-Fletcher-Goldfarb-Shanno”(BFGS) algorithm, which uses a rank two modification of the old $\mathbf{B}$. There are, of course, many other update methods, but it was reported that BFGS method works better with the inexact line search. Hence this method is preferred in practice. In summary, the only difference between the Quasi-Newton method and the Newton method concerns $\mathbf{B}$ and $\mathbf{H}$. Hence the major modifications of the above algorithm concern Step 5. A Quasi-Newton algorithm using BFGS updates can be described as follows:

1. Initialization. Set iteration counter k=0, get initial variable values $\mathbf{x_0}$, calculate the Jacobian (gradient) vector $\mathbf{g_0}$ using $\mathbf{x_0}$, and initialize a symmetric positive definite matrix $\mathbf{B_0}$.
2. Check $\mathbf{g_k}$. If it converges to 0, then STOP.
3. Solve for the search direction $\mathbf{d_k}$, $\mathbf{B_k d_k} = -\mathbf{g_k}$. Alternatively this step can be expressed as $\mathbf{d_k} = -\mathbf{B_k}^{-1}\mathbf{g_k}$.
4. Search for α using a line search and set $\mathbf{x_{k+1}} = \mathbf{x_k} + \alpha \mathbf{d_k}$.
5. Calculate the gradient vector $\mathbf{g_{k+1}}$ using $\mathbf{x_{k+1}}$. Set $\Delta\mathbf{x_k} = \mathbf{x_{k+1}} - \mathbf{x_k}$ and $\Delta\mathbf{g_k} = \mathbf{g_{k+1}} - \mathbf{g_k}$. The BFGS update is:

$$\mathbf{B_{k+1}} = \mathbf{B_k} + \frac{\Delta\mathbf{g_k}\Delta\mathbf{g_k}^T}{\Delta\mathbf{g_k}^T\Delta\mathbf{x_k}} - \frac{\mathbf{B_k}\Delta\mathbf{x_k}\Delta\mathbf{x_k}^T\mathbf{B_k}}{\Delta\mathbf{x_k}^T\mathbf{B_k}\Delta\mathbf{x_k}}$$

6. Set k=k+1. Go to 2.

Before we move on to the optimization with bound constraints, there are some details to be elaborated here.

First of all, if we apply the Sherman-Morrison formula [Chong and Żak, 1996] to $\mathbf{B_{k+1}}$ twice in Step 5, $\mathbf{B_{k+1}^{-1}}$ is readily available and in the next iteration Step 3 is easy to carry out. Nevertheless, we will *not* adopt this strategy because of a minor but important practical issue involved.

Since the whole algorithm depends on the positive definiteness property of the matrix $\mathbf{B}$ (otherwise the search direction will be wrong, and it can take much much longer to find the right direction and the target!), it would be good to keep the positive definiteness during all iterations. But there are two cases where the update in Step 5 can result in a non-positive-definite $\mathbf{B}$:

First, the hereditary positive-definiteness is theoretically guaranteed iff $\Delta\mathbf{g_k}^T\Delta\mathbf{x_k} > 0$ and this condition can be ensured in an exact line search [Gill et al., 1981]. When using an inexact line search, apart from the sufficient function decrease criterion, we should also impose this second condition on it. Thus we cannot use the line search in [Press et al., 1992], instead we use the "modified line search" described in [Dennis and Schnabel, 1983] in Step 4.

Second, even if the hereditary positive-definiteness is theoretically guaranteed, the matrix $\mathbf{B}$ can still lose positive-definiteness during the update due to rounding errors when the matrix is nearly singular, which are not uncommon in practice. Therefore we keep a Cholesky factorization of $\mathbf{B}$ during the iterations: $\mathbf{B} = \mathbf{LDL}^T$ where $\mathbf{L}$ is the lower triangle matrix and $\mathbf{D}$ is a diagonal matrix. The positive definiteness of $\mathbf{B}$ can be guaranteed if the diagonal elements of $\mathbf{D}$ are all positive. If the resulting matrix after a low rank update is theoretically positive definite, there exist algorithms that avoid rounding errors during the updates and ensure that all the diagonal elements of $\mathbf{D}$ are positive [Gill, Golub, Murray and Saunders, 1974; Goldfarb, 1976]. This factorized version of the BFGS updates is the reason why we do not use $\mathbf{B_k^{-1}}$ in Step 3 — because with a Cholesky factorization, the equation $\mathbf{B_k d_k} = -\mathbf{g_k}$ can be solved in $O(N^2)$ time (where N is the number of variables) using forward and backward substitution, and hence $\mathbf{B_k^{-1}}$ is no longer needed. The reader might notice that the BFGS update formula in Step 5 is not convenient if the Cholesky factorization of $\mathbf{B_k}$ is involved. However, using the fact that $\mathbf{B_k}\Delta\mathbf{x_k} = \alpha\mathbf{B_k d_k} = -\alpha\mathbf{g_k}$, we can simplify the formula to:

$$\mathbf{B_{k+1}} = \mathbf{B_k} + \frac{\Delta\mathbf{g_k}\Delta\mathbf{g_k}^T}{\Delta\mathbf{g_k}^T\Delta\mathbf{x_k}} + \frac{\mathbf{g_k}\mathbf{g_k}^T}{\mathbf{g_k}^T\mathbf{d_k}}$$

Note that this involves two rank one modifications, and the coefficients $\frac{1}{\Delta \mathbf{g_k}^T \Delta \mathbf{x_k}} > 0$ and $\frac{1}{\mathbf{g_k}^T \mathbf{d_k}} < 0$ respectively. Hence the first update is a positive rank one update and the second one a negative rank one update. There is a direct rank two modification algorithm [Goldfarb, 1976], but for simplicity we implemented two rank one modifications using the C1 algorithm in [Gill et al., 1974] for the former update and the C2 algorithm, also in [Gill et al., 1974], for the latter one. Note that all these algorithms have $O(N^2)$ complexity.

In summary, we use a factorized version of the Quasi-Newton method to avoid the rounding errors and achieve positive-definiteness of $\mathbf{B}$ during updates. Note that the total complexity of using Cholesky factorization is $O(N^2)$. If we did not use it, the computational cost would still be $O(N^2)$ due to the matrix multiplication. Therefore there is hardly additional expense for computing Cholesky factorization.

Finally, we reach the topic of optimization subject to bound constraints. We adopt basically the same strategy as described in [Gill and Murray, 1976] and [Gill et al., 1981]. It is fairly similar to the above unconstrained optimization method.

First we consider the optimization subject to linear equality constraints $\mathbf{Ax} = \mathbf{b}$. It is an easy problem because it can actually be cast as an unconstrained optimization problem with a reduced dimensionality. A common method to solve this problem is the "projected gradient method", in which the above Quasi-Newton method with BFGS updates remains virtually unchanged. We simply replace the gradient vector $\mathbf{g}$ and the matrix $\mathbf{B}$ by projected versions $\mathbf{Zg}$ and $\mathbf{Z}^T\mathbf{BZ}$ respectively, where $\mathbf{Z}$ is a projection matrix. There are various methods to calculate $\mathbf{Z}$ and usually the orthogonal projection of $\mathbf{A}$ (in the constraints) is taken [Chong and Żak, 1996; Gill et al., 1981]. Particularly if the constraints are bound constraints, it is typically easy to calculate because some variables become constants and do not affect the objective function any more. The projection matrix $\mathbf{Z}$ is thus simply a vector with entries of 1 for "free" (i.e. not in the constraints) variables and 0s otherwise.

Next let us go further into the problem of optimization subject to linear inequality constraints $\mathbf{Ax} \geq \mathbf{b}$. There are several options to solve this kind of problems but we are interested in the one(s) that does not allow variables to take values over the bounds, because in our case the objective function is not defined there. Hence we use the "active set method", which has this essential feature [Gill et al., 1981]. The idea of the "active set method" is to check the search step in each iteration such that, if some variables are about to violate the constraints, these constraints become the "active set" of constraints. Then, in later iterations, use the projected gradient and the projected Hessian (or $\mathbf{B}$ in the Quasi-Newton case) corresponding to the inactive constraints to perform further steps. We will not dig deeply into this method because in our case (i.e. for bound constraints), the task is especially easy. In each iteration in the above Quasi-Newton method with BFGS updates, we simply test whether a search step can cause a variable to go beyond the corresponding bound. If this occurs, we "fix" this variable, i.e. treat it as a constant in later iterations, and use only the "free" variables to carry out the search. Thus the main modification in the above algorithm is in the line search in Step 4. We should use an upper bound for α for all possible variables. The upper bound is $\min(\frac{b_i - x_i}{d_i})$ (where b_i is the bound constraint for the i^{th} variable x_i) if the direction d_i of x_i is an infeasible direction (i.e. if $d_i < 0$). This means that we always calculate the maximal step length that does *not* violate any of the inactive constraints and set this as the upper bound for the trial. Therefore this line search is called "safeguarded line search". This method can be readily extended to the case of two-sided bound constraints, i.e. $\mathbf{u} \geq \mathbf{x} \geq \mathbf{l}$, which is now in the implementation in WEKA.

Last but not least, there is a natural question to be asked regarding our method: "will any 'fixed' variables be released some time? If so, when and how?". The answer is certainly "yes". In our strategy, we only check the possibility of releasing fixed variables when the convergence of the gradient is detected. At that moment, we verify both the first and second order estimates of the Lagrange multipliers of all the fixed variables (where the function implementing the second derivatives are provided by the user). If they are consistent with each other, we regard the second order estimate as a valid one and check whether it is negative. The negativity of a valid Lagrange multiplier indicates non-optimality, hence the corresponding variables can be made "free". If any fixed variables are to be released, then we project the gradient and the

Hessian back to the corresponding higher dimensions, i.e. update the corresponding entries in $\mathbf{g}$ and $\mathbf{B}$ (basically set them to the initial state for these originally "fixed" variables). Nonetheless, if the user does not provide the second derivative,[2] we only use the first order estimate of the Lagrange multiplier.

The above is a description of what we have done for the optimization subject to bound constraints. For completeness, we write down the final algorithm in the following, although it is basically just a repetition of the above description. Note we use the superscript "FREE" to indicate a "projected" vector or matrix below (i.e. they only have entries corresponding to the *free* variables).

1. Initialization. Set iteration counter k=0, get initial variable values $\mathbf{x_0}$, calculate the Jacobian (gradient) vector $\mathbf{g_0}$ using $\mathbf{x_0}$ and compute the Cholesky factorization of a symmetric positive definite matrix $\mathbf{B_0}$ using a lower triangle unit matrix $\mathbf{L_0}$ and a diagonal matrix $\mathbf{D_0}$.

2. Check $\mathbf{g_k}$. If it converges to 0, then test whether any fixed (or bound) variables can be released from their constraints using both first and second order estimates of the Lagrange multipliers.

3. If no variable can be released, then STOP, otherwise release the variables and add corresponding entries in $\mathbf{x_k}^{FREE}$ (set to the bound values), $\mathbf{g_k}^{FREE}$ (set to the gradient values at the bound values), $\mathbf{L_k}^{FREE}$, and $\mathbf{D_k}^{FREE}$ (if the j^{th} variable is to be released, we set l_{jj} and d_{jj} to 1 and the other entries in j^{th} row/column to 0).

4. Solve for the search direction $\mathbf{d_k}^{FREE}$ using backward and forward substitution, $\mathbf{L_k}^{FREE}\mathbf{D_k}^{FREE}(\mathbf{L_k}^{FREE})^T\mathbf{d_k}^{FREE} = -\mathbf{g_k}^{FREE}$.

5. Cast an upper bound on α and search for the best value of α along the direction $\mathbf{d_k}^{FREE}$ using a safeguarded inexact line search with backtracking and polynomial interpolation. Set $\mathbf{x_{k+1}}^{FREE} = \mathbf{x_k}^{FREE} + \alpha\mathbf{d_k}^{FREE}$.

[2] It is often the case because one of the reasons why people use the Quasi-Newton method is that they do not need to provide the Hessian matrix.

6. If any variable is “fixed” at its bound constraint, delete its corresponding entries in $\mathbf{x_k}^{FREE}$, $\mathbf{g_k}^{FREE}$, $\mathbf{L_k}^{FREE}$, and $\mathbf{D_k}^{FREE}$.

7. Calculate the gradient vector $\mathbf{g_{k+1}}^{FREE}$ using $\mathbf{x_{k+1}}^{FREE}$. Set $\Delta\mathbf{x_k}^{FREE} = \mathbf{x_{k+1}}^{FREE} - \mathbf{x_k}^{FREE}$ and $\Delta\mathbf{g_k}^{FREE} = \mathbf{g_{k+1}}^{FREE} - \mathbf{g_k}^{FREE}$. Then the update is:

$$\begin{aligned}\mathbf{L_{k+1}}^{FREE}\mathbf{D_{k+1}}^{FREE}(\mathbf{L_{k+1}}^{FREE})^T &= \mathbf{L_k}^{FREE}\mathbf{D_k}^{FREE}(\mathbf{L_k}^{FREE})^T\\ &+ \frac{\Delta\mathbf{g_k}^{FREE}(\Delta\mathbf{g_k}^{FREE})^T}{(\Delta\mathbf{g_k}^{FREE})^T\Delta\mathbf{x_k}^{FREE}}\\ &+ \frac{\mathbf{g_k}^{FREE}(\mathbf{g_k}^{FREE})^T}{(\mathbf{g_k}^{FREE})^T\mathbf{d_k}^{FREE}}\end{aligned}$$

   We use the aforementioned C1 and C2 algorithms to perform the updates.

8. Set k=k+1. GO TO 2.

Appendix C

Fun with Integrals

This Appendix is about the detailed derivation of the final equations for both TLD and TLDSimple in Chapter 5.

C.1 Integration in TLD

As discussed based on Equations 5.3, 5.4 and the second line of 5.6 in Chapter 5,

$$\begin{aligned} B_{jk} = \int_0^{+\infty}\int_{-\infty}^{+\infty} \Bigg\{ & (2\pi\sigma_k^2)^{-n_j/2} \exp\Big[-\frac{S_{jk}^2 + n_j(\overline{x}_{jk}-\mu_k)^2}{2\sigma_k^2}\Big] \\ & \frac{a_k^{\frac{b_k}{2}}\, 2^{-\frac{b_k+1}{2}}}{\sqrt{(\pi w_k)}\Gamma(b_k/2)} (\sigma_k^2)^{-\frac{b_k+3}{2}} \exp\Big[-\frac{a_k + \frac{(\mu_k-m_k)^2}{w_k}}{2\sigma_k^2}\Big]\Bigg\}\, \mathrm{d}\mu_k\, \mathrm{d}\sigma_k^2. \end{aligned}$$

Re-arranging it, we get

$$\begin{aligned} = \int_0^{+\infty}\int_{-\infty}^{+\infty} \Bigg\{ & \frac{a_k^{\frac{b_k}{2}}}{2^{\frac{b_k+n_j}{2}}\sqrt{(2\pi w_k)}\Gamma(b_k/2)} (\sigma_k^2)^{-\frac{b_k+n_j+3}{2}} \exp(-\frac{a_k}{2\sigma_k^2}) \\ & \exp\Big(-\frac{1}{2w_k\sigma_k^2}[w_k S_{jk}^2 + n_j w_k(\overline{x}_{jk}-\mu_k)^2 + (\mu_k-m_k)^2]\Big)\Bigg\}\, \mathrm{d}\mu_k\, \mathrm{d}\sigma_k^2. \end{aligned}$$

Since

$$[w_k S_{jk}^2 + n_j w_k(\overline{x}_{jk} - \mu_k)^2 + (\mu_k - m_k)^2] = (1 + n_j w_k)\Big[\mu_k - \frac{n_j w_k \overline{x}_{jk} + m_k}{1 + n_j w_k}\Big]^2 + \frac{w_k n_j(\overline{x}_{jk} - m_k)^2 + w_k S_{jk}^2(1 + n_j w_k)}{1 + n_j w_k},$$

we can further re-arrange the above equation as

$$B_{jk} = \int_0^{+\infty}\int_{-\infty}^{+\infty}\Bigg\{\frac{a_k^{\frac{b_k}{2}}}{2^{\frac{b_k+n_j}{2}}\sqrt{(2\pi w_k)}\Gamma(b_k/2)}(\sigma_k^2)^{-\frac{b_k+n_j+3}{2}}\exp(-\frac{a_k}{2\sigma_k^2}) \exp\Big(-\frac{1}{2\sigma_k^2}\big[\frac{n_j(\overline{x}_{jk} - m_k)^2 + S_{jk}^2(1 + n_j w_k)}{1 + n_j w_k}\big]\Big)\exp\big[-\frac{(\mu_k - M_k)^2}{2V_k}\big]\Bigg\}\,\mathrm{d}\mu_k\,\mathrm{d}\sigma_k^2$$

where $M_k = \frac{n_j w_k \overline{x}_{jk} + m_k}{1 + n_j w_k}$ and $V_k = \frac{w_k \sigma_k^2}{1 + n_j w_k}$. Using the identity

$$\int_{-\infty}^{+\infty}(2\pi V_k)^{-\frac{1}{2}}\exp[\frac{(\mu_k - M_k)^2}{2V_k}]\,\mathrm{d}\mu_k = 1,$$

we integrate out μ_k,

$$= \int_0^{+\infty}\Bigg\{\frac{a_k^{\frac{b_k}{2}}}{2^{\frac{b_k+n_j}{2}}\sqrt{(2\pi w_k)}\Gamma(b_k/2)}(\sigma_k^2)^{-\frac{b_k+n_j+3}{2}}\exp(-\frac{a_k}{2\sigma_k^2}) \exp\Big(-\frac{1}{2\sigma_k^2}\big[\frac{n_j(\overline{x}_{jk} - m_k)^2 + S_{jk}^2(1 + n_j w_k)}{1 + n_j w_k}\big]\Big)\sqrt{2\pi\frac{w_k\sigma_k^2}{1 + n_j w_k}}\Bigg\}\,\mathrm{d}\sigma_k^2.$$

Now we set $y = \frac{a_k}{2\sigma_k^2}$ and re-arrange again:

$$B_{jk} = \int_0^{+\infty}\Bigg\{\frac{2a_k^{-(n_j+2)/2}}{\pi^{n_j/2}\sqrt{1 + n_j w_k}\Gamma(b_k/2)}y^{\frac{b_k+n_j+2}{2}}\exp(-\alpha y)\Bigg\}\,\mathrm{d}\sigma_k^2$$

where $\alpha = \big[(1 + n_j w_k)(a_k + S_{jk}^2) + n_j(\overline{x}_{jk} - m_k)^2\big]/\big[a_k(1 + n_j w_k)\big]$. Because

$\sigma_k^2 = \frac{a_k}{2y} \Rightarrow \ \mathrm{d}\sigma_k^2 = -\frac{a_k}{2}y^{-2}\,\mathrm{d}y$, we set $\beta = \frac{b_k+n_j}{2}$ and get

$$\begin{aligned}
&= -\int_{+\infty}^{0}\left\{\frac{a_k^{-n_j/2}}{\pi^{n_j/2}\sqrt{1+n_jw_k}\Gamma(b_k/2)}y^{\beta-1}\exp(-\alpha y)\right\}\mathrm{d}y \\
&= \int_{0}^{+\infty}\left\{\frac{a_k^{-n_j/2}}{\pi^{n_j/2}\sqrt{1+n_jw_k}\Gamma(b_k/2)}y^{\beta-1}\exp(-\alpha y)\right\}\mathrm{d}y
\end{aligned}$$

Since $\Gamma(\beta) = \int_0^{+\infty} e^{-t}t^{\beta-1}\,\mathrm{d}t$, substituting with $t = \alpha y$, we get the well-known identity: $\frac{\Gamma(\beta)}{\alpha^\beta} = \int_0^{+\infty} e^{-\alpha y}y^{\beta-1}\,\mathrm{d}y$ [Artin, 1964]. Hence the solution becomes:

$$\begin{aligned}
B_{jk} &= \frac{a_k^{-n_j/2}\Gamma(\beta)}{\pi^{n_j/2}\sqrt{1+n_jw_k}\alpha^\beta\Gamma(b_k/2)} \\
&= \frac{a_k^{b_k/2}(1+n_jw_k)^{(b_k+n_j-1)/2}\Gamma(b_k+n_j/2)}{\left[(1+n_jw_k)(a_k+S_{jk}^2)+n_j(\overline{x}_{jk}-m_k)^2\right]^{\frac{b_k+n_j}{2}}\pi^{\frac{n_j}{2}}\Gamma(\frac{b_k}{2})}
\end{aligned}$$

This is the formula we got in Equation 5.6.

C.2 Integration in TLDSimple

In TLDSimple, we regard σ_k^2 as fixed and estimate it directly from the data. Therefore in Equation 5.2 we plug in a Gaussian-Gaussian formulation, that is, the $\overline{x}_{jk}$ of each bag has the sampling distribution of a Gaussian, $N(\mu_k, \frac{\sigma_k^2}{n_j})$ (according to Central Limit Theorem), and μ_k further follows a Gaussian parameterized by m_k and w_k, $N(m_k, w_k)$. σ_k^2 is now fixed. Hence Equation 5.6 now becomes:

$$B_{jk} = \int_{-\infty}^{+\infty}\left\{\frac{1}{\sqrt{2\pi\frac{\sigma_k^2}{n_j}}}\exp\left[-\frac{(\overline{x}_{jk}-\mu_k)^2}{2(\frac{\sigma_k^2}{n_j})}\right]\frac{1}{\sqrt{(2\pi w_k)}}\exp\left[-\frac{(\mu_k-m_k)^2}{2w_k}\right]\right\}\mathrm{d}\mu_k.$$

Re-arranging it, we have

$$\begin{aligned}
&= \int_{-\infty}^{+\infty}\left\{\frac{1}{\sqrt{(2\pi V_k)}}\exp\left[-\frac{(\mu_k-M_k)^2}{2V_k}\right]\right. \\
&\qquad\left.\left(2\pi\frac{w_kn_j+\sigma_k^2}{n_j}\right)^{-1/2}\exp\left[\frac{-n_j(\overline{x}_{jk}-m_k)^2}{2(w_kn_j+\sigma_k^2)}\right]\right\}\mathrm{d}\mu_k
\end{aligned}$$

where $M_k = \frac{n_j w_k \overline{x}_{jk} + m_k \sigma_k^2}{\sigma_k^2 + n_j w_k}$ and $V_k = \frac{w_k \sigma_k^2}{\sigma_k^2 + n_j w_k}$. With the identity

$$\int_{-\infty}^{+\infty} \frac{1}{\sqrt{(2\pi V_k)}} \exp\Big[-\frac{(\mu_k - M_k)^2}{2V_k}\Big]\,\mathrm{d}\mu_k = 1,$$

we integrate out μ_k and get

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

This is Equation 5.9 from Chapter 5.

Appendix D

Comments on EM-DD

EM-DD [Zhang and Goldman, 2002] was proposed to overcome the difficulty of the optimization problem required to find the maximum likelihood estimate (MLE) of the instance-level class probability parameters in Diverse Density (DD) [Maron, 1998]. Note that there are two approximate generative models proposed in DD to construct the bag-level class probability from the instance-level ones — the noisy-or and the most-likely-cause model. In the noisy-or model, there is no difficulty in optimizing the objective (log-likelihood) function while in the most-likely-cause model, the objective function is not differentiable because of the "maximum" functions involved. DD used the "softmax" function to approximate the maximum function in order to facilitate gradient-based optimization, which is a standard way to solve non-differentiable optimization problems. EM-DD, on the other hand, claims to use an application of the EM algorithm [Dempster et al., 1977] to circumvent the difficulty. Therefore EM-DD provides no improvement on the modeling process in DD, only on the optimization process. Since DD uses the standard way to treat non-differentiable optimization problems [Lemaréchal, 1989] and was supposed to find the MLE, why can EM-DD be such an improvement? Besides, we have not found any case in the literature that EM can simplify a non-differentiable log-likelihood function in conjunction with a gradient-based optimization method (i.e. a Newton-type method) in the "M-step". Can the standard EM algorithm be applied to a non-differentiable log-likelihood function? These questions lead us to be skeptical about the validity of EM-DD.

In the following we first analyze the log-likelihood function that EM-DD aims to maximize, then we present the EM-DD algorithm and an illustrative example to see whether it can work on this function. Finally we point out a mistake in the "proof" for EM-DD. We can also show that in general the monotonicity of EM-DD cannot be proved, thus theoretically EM-DD is not guaranteed to work.

D.1 The Log-likelihood Function

EM-DD is based on the log-likelihood function constructed with the most-likely-cause model (Equation 7.3 in Chapter 7):

$$\begin{aligned} L = & \sum_B [Y \log(max_{b \in m}\{Pr(Y=1|x_b)\}) + \\ & (1-Y)\log(1 - max_{b \in m}\{Pr(Y=1|x_b)\})] \\ = & \sum_B [Y \log(max_{b \in m} Pr(Y=1|x_b)) + (1-Y)\log(min_{b \in m} Pr(Y=0|x_b))] \end{aligned} \tag{D.1}$$

The parameter vector β determines our estimate of $Pr(Y = 1|x_b)$, and we seek the value of β that maximizes L, i.e. β_{MLE}. Given a certain value of β, we select one instance from each bag (the "most-likely-cause" instance) to construct the log-likelihood. In the process of searching for β_{MLE}, when the parameter value changes, the "most-likely-cause" instance to be selected may also change. Thus, the log-likelihood function may suddenly change forms when the parameter value changes. More specifically, if we arbitrarily pick up one instance from each bag and construct the log-likelihood function, we have one possible log-likelihood function — we call it one "component function". If we change the instance in one bag, we obtain another, different (unless the changed instance is identical to the old one) component function. Obviously, if there are s_1 instances in the 1^{st} bag, s_2 instances in the 2^{nd} bag, …, s_{m+n} instances in the $(m+n)^{th}$ bag, then there are $s_1 \times s_2 \times \cdots \times s_{m+n}$ component functions available (where m and n are the number

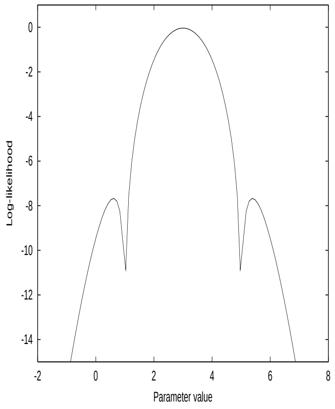


Figure D.1: A possible component function in one dimension.

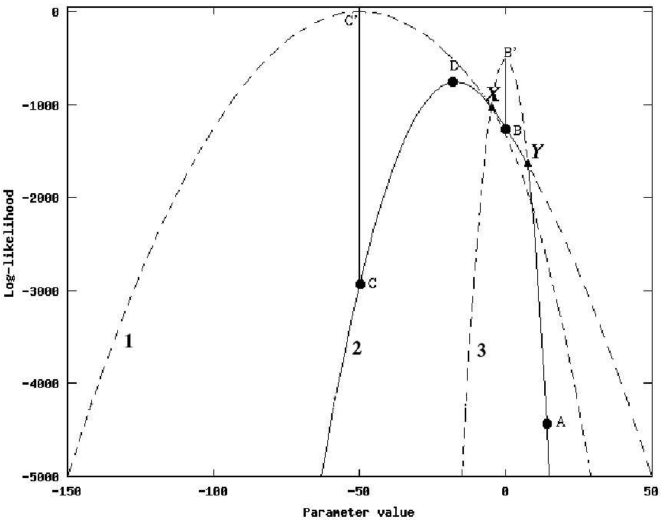


Figure D.2: An illustrative example of the log-likelihood function in DD using the most-likely-cause model.

of positive and negative bags respectively). And the true log-likelihood function is constructed using *some* of these component functions.[1] When the parameter value falls into one range in the domain of β, the log-likelihood function is in the form of a certain component function. And if it falls into another range, then it becomes another component function. Although the true log-likelihood function changes its form for different domains of β, it is continuous at the point when the form of the function changes from one component function to another because of the Euclidean distance (L_2 norm) used in the radial (or Gaussian-like) formulation of $Pr(Y|X)$, but it is no longer differentiable at that point.

In Figure D.1 we show the shape of a part of one possible component function in one dimension, i.e. we only have one attribute and fix the value of the "scaling parameter" in DD. Thus the only variable here is the "point parameter". Note that there are three local maxima in this function, and the function is not continuous because the log-likelihood is undefined in two locations (actually the parameter value in any location is equal to the attribute value of an instance in a negative bag). This is due to the radial formulation of $Pr(Y|X)$ in DD. The usual way in both DD and

[1] Note that not all of the component functions are used because some instances (from different bags) will never be picked up simultaneously for any value of β.

EM-DD to tackle this problem is to search for the parameter values using (different) multiple starts, in the hope that one start point can lead to the global maximum of the function. From now on, we assume that we can always find the (global) maximum for each component function, perhaps using multiple starts. Regardless of the local maxima, the shape of a component function is roughly quadratic. This is reasonable because at least for the parts of the function constructed using only the positive instances (in this appendix we say an instance in a positive bag a "positive instance" and an instance in a negative bag a "negative instance"), it is exactly quadratic. The negative instances only make the function discontinuous, as shown in the function's shoulders in Figure D.1 (actually the figure does not show clearly the discontinuity of the function — there should be no minimum on the shoulders, instead the function value goes to $-\infty$ there).

If we ignore the (small) local maxima in each of the component functions, we can illustrate (a part of) the true likelihood function using curves like those in Figure D.2. Note that this figure does not rigorously describe the situation of maximizing the log-likelihood in Equation D.1. It only serves an illustration. However, it does give us an idea when EM-DD can work and when it cannot. Although some details may not be accurate (like the coordinates or the exact shape of the component functions), these factors do not affect the key ideas of the illustration.

There are three component functions in the plot, denoted by 1, 2 and 3. The dotted lines are the part of the component functions *not* used in the true log-likelihood function. The solid line plots the true function. Again we only plot, in one dimension, the "point parameter" against the log-likelihood. The shapes of the component functions are different because we also incorporate a fixed value of the "scaling parameter" for each one of the component functions (i.e. different component functions have different "scaling parameter" values). Therefore, we simplify the optimization procedure as a "steepest descent" method where we first fix the "scaling parameter" and search for the best "point parameter", then fix the "point parameter" and search for the optimal "scaling parameter". In Figure D.2, we do not show how to search for the value of "scaling parameter". We assume that we are "given" the optimal “scaling parameter” value every time we start searching for the “point parameter”. We also assume that the “steepest descent” procedure does equally well as the Quasi-Newton method used in DD and EM-DD on this simple problem. We need to make all these assumptions to simplify the situation and enable us to see the essence of this (extremely complicated) optimization problem.

Note that the true function value can be *smaller* than the values of component function. To see why, let us look back to the Equation D.1. Given a fixed parameter β_f, picking an instance with the maximal value of $Pr(Y = 1|X, \beta_f)$ in a positive bag must result in a greater log-likelihood value than picking up another instances in the same bag. On the contrary, selecting an instance with the maximal value of $Pr(Y = 1|X, \beta_f)$ (i.e. the minimal value of $Pr(Y = 0|X, \beta_f)$) in a *negative* bag has to result in a *smaller* log-likelihood value than selecting other instances in the same bag. Therefore the true log-likelihood function is often *not* the one with the greatest value among all the component functions, as illustrated in Figure D.2, as long as there is more than one instance in each negative bag.

In this example, the true function shifts between the components twice, and the shifting points are indicated by small triangles and “***X***” and “***Y***” respectively. At both points the log-likelihood function is still increasing but the newly-shifted component function value is less than the value of the old component function. Shifting between components means that we should change the instances in each bag that construct the log-likelihood function. In the new component function, a new set of instances, one from each bag, are picked up. Obviously this true function is non-differentiable at the two points when it shifts components, but it has a local maximum at point D.

D.2 The EM-DD Algorithm

Now we briefly sketch the EM-DD algorithm [Zhang and Goldman, 2002]. EM-DD iterates with initial values of β, say β_0, and the initial log-likelihood as

$$\begin{aligned} L_0 = &\sum_{1\le i\le n} \log[max_{b\in m}\{Pr(Y=1|x_b,\beta_0)\}]+ \\ &\sum_{1\le j\le m} \log[1-max_{b\in m}\{Pr(Y=1|x_b,\beta_0)\}]. \end{aligned}$$

Then it cycles between the E-step and M-step as follows until convergence (Suppose this is the p^{th} iteration):

- E-step: find the instance in each bag that has the greatest $Pr(Y|X,\beta_p)$, say, the z_i^{th} instance in the i^{th} bag.

- M-step: search for

$$\begin{aligned} \beta_{p+1} = argmax\{ &\sum_{1\le i\le n} \log[Pr(Y=1|x_{iz_i},\beta)]+ \\ &\sum_{1\le j\le m} \log[(1-Pr(Y=1|x_{jz_j},\beta))]\} \end{aligned}$$

  where there are n positive bags and m negative bags. And then compute

$$\begin{aligned} L_{p+1} = &\sum_{1\le i\le n} \log[max_{b\in m}\{Pr(Y=1|x_b,\beta_{p+1})\}]+ \\ &\sum_{1\le j\le m} \log[1-max_{b\in m}\{Pr(Y=1|x_b,\beta_{p+1})\}] \end{aligned}$$

The convergence test is performed before each E-step (or after each M-step) via the comparison of the values of the log-likelihood function L_p and L_{p+1}.

To give an illustration, we first apply the above algorithm to the artificial example shown in Figure D.2 and see what it could find given “A” as the start value for β. This example is deliberately set up so that we may see both cases in which EM-

DD can work (in the first iteration) and cannot work (in the second iteration). As mentioned before we assume the search procedure used in the M-step is "steepest descent" — one member of the gradient descent family. Although EM-DD used a Quasi-Newton method, this does not matter much in this simple situation. Note that the "M-step" in EM-DD corresponds to searching for the maximum in *one component* function because the instance to be selected in each bag is fixed, which means it often goes *above* the true log-likelihood function in "M-step".

In the E-step in the first iteration, EM-DD selects the set of instances as required. This is to say it finds the correct component function — the curve with the solid line. Then it calculates L_0 as the function value at the point A. In the M-step it finds point B' as the maximum of Component 3 (assuming it also finds the optimal "scaling parameters" in the M-step, which determine the shape of next component in Figure D.2). When it computes the log-likelihood L_1, it will necessarily "return" to the true log-likelihood function. In other words, it picks up the new set of instances according to β_1 (the X-axis coordinate of B/B'). As a result L_1 is the value of the log-likelihood function at point B.

In the second iteration, EM-DD first compares L_1 and L_0, i.e. the function values at A and B. In this case, it indeed finds a better value of the parameter, so it continues. In the E-step it picks an instance in each bag according to β_1. The corresponding new component function is Component 1. In the M-step it will find the maximum of Component 1 at point C'. Then it calculates L_2 as the true function value. This is actually point C on Component 2.

In the third iteration, EM-DD first compares L_2 and L_1, i.e. the function values at B and C. However, this time it finds that it cannot increase the function value so the algorithm stops and point B is returned as the solution, i.e. EM-DD will not be able to find point D which is the true maximum of the log-likelihood function. If it kept searching, it would have found D. Nonetheless, to do this, it must break the convergence test, which is a crucial part of the proof of convergence for EM-DD [Zhang and Goldman, 2002]. Without this convergence test EM-DD is not an application of EM but simply a stochastic searching method for a combinatoric optimization problem.

Therefore, even if we assume that the global maximum in each component function can be found, EM-DD cannot find the maximum of the log-likelihood function. Note that we have simplified this example a lot — the objective function is concave and in one dimension in this case. In reality, since there are many more dimensions (typically EM-DD searches for 2×166 parameters simultaneously on the Musk datasets), the situation is much more complicated than the above example. For instance, saddle points can occur, which is a case that EM cannot deal with anyway [McLachlan and Krishnan, 1996].

Note that in the above example, we required EM-DD to start from point A. With a different start point for the search for β, it may find the maximum point D. Indeed, we observed that EM-DD depends heavily on multiple start points, not only for searching for the global maximum but also for improving the chances to find just a local maximum. In other words, it really relies on good luck rather than strong theoretical justifications.

D.3 Theoretical Considerations

In spite of the above counter-example, it is not sufficient to convince ourselves that EM-DD is not a valid algorithm because it was proved in [Zhang and Goldman, 2002] that this algorithm will converge to a local maximum. This proof is analogous to that in the EM algorithm. However, it turns out that the most important part was missed.

The Expectation-Maximization (EM) algorithm [Dempster et al., 1977] was proposed to facilitate the maximum likelihood method when “unobservable” data is involved in the log-likelihood function. It is discussed in detail in a variety of articles or books [Bilmes, 1997; McLachlan and Krishnan, 1996]. Since the M-step necessarily increases the log-likelihood function, the key for proving the monotonicity of the EM algorithm is to prove that the E-step can also increase the log-likelihood. The property of an increase in the log-likelihood function in the E-step is a consequence of Jensen's inequality and the concavity of the logarithmic function [Dempster et al., 1977; McLachlan and Krishnan, 1996]. That is why EM can also be viewed as a "Maximum-Maximum" procedure [McLachlan and Krishnan, 1996; Hastie et al., 2001]. Nonetheless, [Zhang and Goldman, 2002] does not provide proof of the increase of the log-likelihood function (in Equation D.1) in the E-step at all. Instead it uses the convergence test (of terminating the algorithm if $L_p \geq L_{p+1}$) to prevent the log-likelihood from decreasing. Note that in standard EM, since the E-step also increases the log-likelihood, the convergence test is only to test whether $L_p = L_{p+1}$, involving no "$>$" sign.

The reason why the proof used in the standard EM algorithm does not apply to EM-DD is due to the special property of the "unobservable" data. In EM-DD, the unobservable data is z_i, an index for the i^{th} bag that indicates which instance should be used in the log-likelihood function. This variable z_i is not quite "unobservable" in this case because for each value of β, it is fixed (i.e. no probability distribution is needed) and observable in the data, although for different values of β its value also changes. Therefore if one insists on regarding it as a latent variable in EM, then given a certain parameter value β_p, the probability of z_i is

$$Pr(z_i|\beta_p) = \begin{cases} 1 & \text{if } z_i = argmaxPr(Y = 1|x_{iz_i}, \beta_p) \\ 0 & \text{otherwise} \end{cases}$$

This probability function is very unusual, and still involves a max function, which is not smooth, and thus the proof in EM cannot apply to the expected log-likelihood function in the E-step in EM-DD.

As a matter of fact, as shown in Section D.2, in the E-step of EM-DD, the log-likelihood is very likely to *decrease*, in which case the algorithm has to stop. Note that in Equation D.1, in a negative bag, $1 - max_{b \in m}\{Pr(Y = 1|x_b)\}$ is equivalent to $min_{b \in m}\{1 - Pr(Y = 1|x_b)\} = min_{b \in m}\{Pr(Y = 0|x_b)\}$. Hence in the E-step, changing any negative instances to construct a new log-likelihood function will always *decrease* the log-likelihood function. In the extreme, if in one E-step, the positive instances involved in the current log-likelihood function remain unchanged, but some negative instances are changed, then the new log-likelihood is guaranteed to be lower than the current one. In that case, it may be premature to halt the algorithm, as shown in the example in Section D.2. Therefore, unlike EM, the monotonicity of the E-step in EM-DD cannot be proved in general, which is the major theoretical flaw in EM-DD.

The fact that the log-likelihood fails to increase "after the first several iterations" for EM-DD [Zhang and Goldman, 2002] is probably due to a decrease in the E-step. Moreover, it was also observed that "it is often beneficial to allow NLDD (the negative log-likelihood of Diverse Density) to increase slightly" [Zhang and Goldman, 2002]. We believe this is not solely because of local maxima (or minima for the negative log-likelihood) — it may also allow the algorithm to keep searching, where it would otherwise fail. Indeed, without the convergence requirement of EM (that EM-DD cannot achieve), we can develop an algorithm that is guaranteed to find the solution of the MLE — we simply search for the global maximum in each of the component functions, either in a systematic (say, branch-and-bound) or stochastic manner, and pick up the parameter with the highest "true" log-likelihood. This amounts to searching in all the component functions involved in the log-likelihood function. However this has nothing to do with EM. And the computational expense varies greatly from case to case. The worst-case cost could be very high.

The problem with EM-DD lies in the objective of using a normal gradient-based EM to solve a non-differentiable optimization problem. Non-differentiable optimization has been a hot research topic in the optimization domain for some time [Lemaréchal, 1989]. One of the methods to deal with non-differentiable optimization problem is to transform the objective function into a differentiable function. The method used in DD to substitute the "max" function with the "softmax" function is of this kind. Although the "softmax" does not accurately transform the function, it approximates the true log-likelihood function precisely enough. In the case shown in Figure D.2, it will approximate the true function using a differentiable function, hence this differentiable function will look similar to the true log-likelihood function and has a maximum point close enough to point D. Using a normal Newton-type method we can easily find this point.

In summary, because DD uses a sound maximization procedure whereas EM-DD's approach may not find an MLE, we are inclined to believe the statement in [Maron, 1998] that DD with the most-likely-cause model actually performs worse than with the noisy-or model on the Musk datasets, and we are skeptical about the good results reported for EM-DD [Zhang and Goldman, 2002] (especially considering that there are also problems with the evaluation procedure used in [Zhang and Goldman, 2002]).

Bibliography

Ahrens, J. and Dieter, U. [1974]. Computer methods for sampling from Gamma, Beta, Poisson and Binomial distributions. *Computing, (12),* 223–246.

Ahrens, J., Kohrt, K. and Dieter, U. [1983]. Algorithm 599: sampling from Gamma and Poisson distributions. *ACM Transactions on Mathematical Software, 9(2),* 255–257.

Artin, E. [1964]. *The Gamma Function*. New York, NY: Holt, Rinehart and Winston. Translated by M. Butler.

Auer, P. [1997]. On learning from multiple instance examples: empirical evaluation of a theoretical approach. In *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 21–29). San Francisco, CA: Morgan Kaufmann.

Bilmes, J. [1997]. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, University of Berkeley.

Blake, C. and Merz, C. [1998]. UCI repository of machine learning databases.

Blum, A. and Kalai, A. [1998]. A note on learning from multiple-instance examples. *Machine Learning, 30(1),* 23–30.

Bratley, P., Fox, B. and Schrage, L. [1983]. *A Guide to Simulation*. New York, NY: Springer-Verlag.

Breiman, L. [1996]. Bagging predictors. *Machine Learning, 24(2)*, 123–140.

le Cessie, S. and van Houwelingen, J. [1992]. Ridge estimators in logistic regression. *Applied Statistics, 41(1)*, 191–201.

Chevaleyre, Y. and Zucker, J.-D. [2000]. Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. Application to the mutagenesis problem. Internal Report, University of Paris 6.

Chevaleyre, Y. and Zucker, J.-D. [2001]. A framework for learning rules from multiple instance data. In *Proceedings of the Twelveth European Conference on Machine Learning* (pp. 49–60). Berlin: Springer-Verlag.

Chong, E. and Żak, S. [1996]. *An Introduction to Optimization*. New York, NY: John Wiley & Sons, Inc.

Cohen, W. [1995]. Fast effective rule induction. In *Proceedings of the Twelveth International Conference on Machine Learning* (pp. 115–123). San Francisco, CA: Morgan Kaufmann.

Dempster, A., Laird, N. and Rubin, D. [1977]. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistics Society, Series B, 39(1)*, 1–38.

Dennis, J. and Schnabel, R. [1983]. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

Devroye, L., Györfi, L. and Lugosi, G. [1996]. *A Probabilistic Theory of Pattern Recognition*. New York, NY: Springer-Verlag.

Dietterich, T. and Bakiri, G. [1995]. Solving multiclass learning problems via error-correcting output codes. *Journal Artificial Intelligence Research, 2*, 263–286.

Dietterich, T., Lathrop, R. and Lozano-Pérez, T. [1997]. Solving the multiple-instance problem with the axis-parallel rectangles. *Artificial Intelligence, 89(1-2)*, 31–71.

Frank, E. and Witten, I. [1998]. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 144–151). San Francisco, CA: Morgan Kaufmann.

Frank, E. and Witten, I. [1999]. Making better use of global discretization. In *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 115–123). San Francisco, CA: Morgan Kaufmann.

Frank, E. and Xu, X. [2003]. Applying propositional learning algorithms to multi-instance data. Working Paper 06/03, Department of Computer Science, University of Waikato, New Zealand.

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

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

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

Gill, P., Golub, G., Murray, W. and Saunders, M. [1974]. Methods for modifying matrix factorizations. *Mathematics of Computation, 28(126),* 505–535.

Gill, P. and Murray, W. [1976]. Minimization subject to bounds on the variables. Technical Report NPL Report NAC-72, National Physical Laboratory.

Gill, P., Murray, W. and Wright, M. [1981]. *Practical Optimization*. London: Academic Press.

Goldfarb, D. [1976]. Factorized variable metric methods for unconstrained optimization. *Mathematics of Computation, 30(136),* 796–811.

Hastie, T., Tibshirani, R. and Friedman, J. [2001]. *The Elements of Statistical Learning : Data mining, Inference, and Prediction*. New York, NY: Springer-Verlag.

John, G. and Langley, P. [1995]. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence* (pp. 338–345). San Mateo, CA: Morgan Kaufmann.

Lemaréchal, C. [1989]. Nondifferentiable optimization. In Nemhauser, R. Kan and Todd (Eds.), *Optimization*, Volume 1 of *Handbooks in Operations Research and Management Science* chapter VII, (pp. 529–569). Amsterdam: North-Holland.

Long, P. and Tan, L. [1998]. PAC learning axis-aligned rectangles with respect to product distribution from multiple-instance examples. *Machine Learning, 30(1),* 7–21.

Maritz, J. and Lwin, T. [1989]. *Empirical Bayes Methods* (2 Ed.). London: Chapman and Hall.

Maron, O. [1998]. *Learning from Ambiguity*. PhD thesis, Massachusetts Institute of Technology, United States.

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

McLachlan, G. [1992]. *Discriminant Analysis and Statistical Pattern Recognition*. New York, NY: John Wiley & Sons, Inc.

McLachlan, G. and Krishnan, T. [1996]. *The EM Algorithm and Extensions*. New York, NY: John Wiley & Sons, Inc.

Minh, D. [1988]. Generating Gamma variates. *ACM Transactions on Mathematical Software, 4(3),* 261–266.

von Mises, R. [1943]. On the correct use of Bayes' formula. *The Annals of Mathematical Statistics*, *13*, 156–165.

Nadeau, C. and Bengio, Y. [1999]. Inference for the generalization error. In *Advanced in Neural Information Processing Systems*, Volume 12 (pp. 307–313). Cambridge, MA: MIT Press.

O'Hagan, A. [1994]. *Bayesian Inference*, Volume 2B of *Kendall's Advanced Theory of Statistics*. London: Edward Arnold.

Platt, J. [1998]. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges and A. Smola (Eds.), *Advances in Kernel Methods—Support Vector Learning*. Cambridge, MA: MIT Press.

Press, W., Teukolsky, S., Vetterling, W. and Flannery, B. [1992]. *Numerical Recipes in C: The Art of Scientific Computing* (2 Ed.). Cambridge, England: Cambridge University Press.

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

Ramon, J. and Raedt, L. D. [2000]. Multi instance neural networks. In *Attribute-Value and Relational Learning: Crossing the Boundaries*. Workshop at the Seventeenth International Conference on Machine Learning.

Ruffo, G. [2001]. *Learning Single and Multiple Instance Decision Trees for Computer Security Applications*. PhD thesis, Universita di Torino, Italy.

Srinivasan, A., Muggleton, S., King, R. and Sternberg, M. [1994]. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the Fourth International Inductive Logic Programming Workshop* (pp. 161–174).

Stuart, A., Ord, J. and Arnold, S. [1999]. *Classical Inference and the Linear Model*, Volume 2A of *Kendall's Advanced Theory of Statistics*. London: Arnold.

Vapnik, V. [2000]. *The Nature of Statistical Learning Theory*. New York, NY: Springer-Verlag.

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

Wang, Y. and Witten, I. [2002]. Modeling for optimal probability prediction. In *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 650–657). San Francisco, CA: Morgan Kaufmann.

Weidmann, N. [2003]. Two-level classification for generalized multi-instance data. Master's thesis, Albert-Ludwigs-Universität Freiburg, Germany.

Weidmann, N., Frank, E. and Pfahringer, B. [2003]. A two-level learning method for generalized multi-instance problems. In *Proceedings of the Fourteenth European Conference on Machine Learning*. To be published.

Witten, I. and Frank, E. [1999]. *Data Mining: practical machine learning tools and techniques with Java implementations*. San Francisco, CA: Morgan Kaufmann.

Zhang, Q. and Goldman, S. [2002]. EM-DD: An improved multiple-instance learning technique. In *Proceedings of the 2001 Neural Information Processing Systems (NIPS) Conference* (pp. 1073–1080). Cambridge, MA: MIT Press.

Zhang, Q., Goldman, S., Yu, W. and Fritts, J. [2002]. Content-based image retrieval using multiple-instance learning. In *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 682–689). San Francisco, CA: Morgan Kaufmann.