AN ABSTRACT OF THE THESIS OF

Dragos Dorin Margineantu for the degree of Doctor of Philosophy in Computer Science presented on September 21, 2001.

Title:

Methods for Cost-Sensitive Learning.

Signature redacted for privacy.

Abstract approved:

Thomas G. Dietterich

Many approaches for achieving intelligent behavior of automated (computer) systems involve components that learn from past experience. This dissertation studies computational methods for learning from examples, for classification and for decision making, when the decisions have different non-zero costs associated with them. Many practical applications of learning algorithms, including transaction monitoring, fraud detection, intrusion detection, and medical diagnosis, have such non-uniform costs, and there is a great need for new methods that can handle them.

This dissertation discusses two approaches to cost-sensitive classification: input data weighting and conditional density estimation. The first method assigns a weight to each training example in order to force the learning algorithm (which is otherwise unchanged) to pay more attention to examples with higher misclassification costs. The dissertation discusses several different weighting methods and concludes that a method that gives higher weight to examples from rarer classes works quite well. Another algorithm that gave good results was a wrapper method that applies Powell's gradient-free algorithm to optimize the input weights.

The second approach to cost-sensitive classification is conditional density estimation. In this approach, the output of the learning algorithm is a classifier that estimates, for a new data point, the probability that it belongs to each of the classes. These probability estimates can be combined with a cost matrix to make decisions that minimize the expected cost. The dissertation presents a new algorithm, bagged lazy option trees (B-LOTs), that gives better probability estimates than any previous method based on decision trees.

In order to evaluate cost-sensitive classification methods, appropriate statistical methods are needed. The dissertation presents two new statistical procedures: BCost provides a confidence interval on the expected cost of a classifier, and BDeltaCost provides a confidence interval on the difference in expected costs of two classifiers. These methods are applied to a large set of experimental studies to evaluate and compare the cost-sensitive methods presented in this dissertation.

Finally, the dissertation describes the application of the B-LOTs to a problem of predicting the stability of river channels. In this study, B-LOTs were shown to be superior to other methods in cases where the classes have very different frequencies—a situation that arises frequently in cost-sensitive classification problems.

Methods for Cost-Sensitive Learning

by

Dragos Dorin Margineantu

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed September 21, 2001
Commencement June 2002

ACKNOWLEDGEMENTS

I am indebted to many people for helping me, directly or indirectly, to complete this dissertation.

First, and foremost is my advisor, Tom Dietterich. He has always known how to guide me to understand what are the important research questions, and to pursue, or postpone, the study of new ideas. Tom's profound way of thinking has led to technical contributions to my work and to this dissertation. My hope is that at least a very small portion of his skills and deep knowledge have rubbed off on me during my graduate studies. Throughout these years he has been a model for me both as a researcher and as a person.

I would like to thank Prasad Tadepalli for his comments and suggestions that led to major improvements in this dissertation, for teaching me fundamental AI concepts, for teaching me to be rigorous. He has been very influential in my thinking. I owe special debts to Bruce D'Ambrosio, Curtis Cook, and Charles Brunner for their observations on this dissertation, for always being available for discussions, and for their kindness. I would also like to address special thanks to all of the Computer Science Faculty at Oregon State University. Together with my committee members, they provided the perfect environment for my graduate studies. Professors Paul Cull, Bella Bose, and Michael Quinn have taught me fundamental concepts in the theory of computing, algorithms, and parallel processing. Professors Walter Rudd, Tim Budd, Cherri Pancake and Margaret Burnett were always available for questions and discussions in the areas of computer systems, programming languages, and compilers. Professors Gregg Rothermel, Toshimi Minoura, and Jon Herlocker have provided me personal advice. Meeting, and talking to them (sometimes late at night in the corridors of Dearborn Hall) has always been a real pleasure.

In the Department of Computer Science at Oregon State I was lucky to be a colleague of Bill Langford, whose insightful comments have been influential in many occasions for my thinking. Many thanks to Avis Ng, Dan Forrest, Valentina Zubek, Xin Wang, Ray Liere, Chandra Reddy, Wei Zhang, and Adam Ashenfelter for their help, comments, fruitful discussions, and for being great colleagues. I also want to address special thanks to the computing systems support team of the Department of Computer Science at Oregon State led by Jalal Haddad, for providing a great environment for work.

I would like to thank Gabriel Somlo, Andrei Gaspar, Stephanie Moret, Daniela Crivianu-Gaita, and Florin Miclea for their willingness to collaborate in different research projects and for sharing their ideas, helping, and providing useful comments at various moments during the last seven years.

I am grateful to Foster Provost, Pedro Domingos, Rob Holte, Ron Kohavi, Leo Breiman, Chris Drummond, Adrian Silvescu, Andrew Ng, Ion Muslea, Charles Elkan, Raymond Mooney, Janyce Wiebe, Nils Nilsson, and Gary Weiss for sharing at various moments during my graduate studies their thoughts and comments on my work, and for their suggestions for conducting my research.

I would like to thank the American Association for Artificial Intelligence (AAAI) for their financial support of my participation at four National Conferences on Artificial Intelligence, and to the National Science Foundation (NSF), and the Office of Naval Research (ONR) for supporting parts of my reseach.

The roots of this thesis can be traced to my years in graduate school at Oregon State University, but I consider it as a continuation of many years of education.

I would like to take this opportunity to direct my gratitude to Florea Piroi and Margareta Marina — my dearest teachers, two people who have mentored me to aim at high standards and be rigorous, and have inspired my passion for mathematics. Dorel Mihet, Mihai Mogosan, Tiberiu Supercean, Gabriela Oprea, Gheorghe Eckstein, Titu Andreescu, Dorin Manz, Ioan Dobosan and Marinel Serban are the teachers whom I owe the most for mentoring my first steps in mathematics and programming.

I am grateful to Ioan Jurca for being the first person who sat down with me in front of a computer to win me into Computer Science and Engineering. Later, Stefan Holban, Radu Fantaziu, Nicolae Szirbik — distinguished AI researchers and professors, were the first people to nurture my interest for artificial intelligence. Thank you!

I would like to thank Horia Ciocarlie, Vladimir Cretu, Peter Eles, Mircea Vladutiu, Marian Boldea, Dumitru Daba, Stefan Matei, Octavian Lipovan, and Corin Tamasdan — computer science, mathematics, physics, and engineering professors I had during my college years, whom I will remember with distinct pleasure.

Four people whom I deeply admire and who represent models for any researcher and teacher, are Professors Ion Boldea, Zoe Dumitrescu-Busulenga, Bernhard Rothenstein, and Constantin Caloghera. I want to acknowledge their profound influence on the way I am thinking today.

Brad Bauer, and Gene, Twila, and Marguerite Eakin have been the first people I met when I came to Oregon in 1995. Together with them, I made my first steps on the OSU campus and they have remained good friends during all these years.

The unique natural beauties of Oregon have, on many occasions, provided the best environment for evening walks, weekend escapes, or short vacations. Rosemary Armore is the friend who has noticed my pleasure to watch and hear the Ocean and has been many times our host on the Oregon coast. Research ideas, some drafts of my papers, and the first draft of this dissertation have their roots in her home, where the quiet ambiance was disrupted only by the sound of the waves. Thank you!

I would like to thank all the roommates I had during my graduate school studies, Andra and Nicu Vulpanovici, Angela and Catalin Doneanu, Ovidiu Iancu, and Dan Onu for being kind and for never protesting my coming home late at night. The memory of Nicu Vulpanovici, his kindness and generous smile are things I will never forget.

I am truly indebted to Paul Oprisan for his generosity and availability in helping out on many technical or non-technical problems, and for sharing his LaTex expertise in the editing process of this dissertation. I am also thankful to Bader AlMohammad for all his advices and for his help.

Many thanks to Anthony Armore, Manfred Dittrich, Aristotel Popescu, Corina Constantinescu, Voichita and Ioan Dumitriu, Geta and Ivan Farkas, George and Cleo Hirsovescu, Julian and Veronica Mart, Dorina and Adrian Avram, for the special moments we had together during these years.

Probably most of this work would not have been possible without the lifelong friendship and support of Nicu Fantana, and the thoughtfulness and generosity of Lucian Chis. They are my best friends.

My wife, Daciana, was always there when I needed a shoulder to lean on. I thank her for her enduring patience, and for her understanding.

I would like to use this opportunity to praise the memory of my grandparents. I hope to be able to always carry along their spirit of leniency, dignity, and altruism.

Above all, I owe everything to my parents. Let me thank my mother, the most generous person I have known, for encouraging me to believe that the world has no limits, and to thank my father for teaching me that sometimes it is good to know that this is not the case.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Figure | Page

LIST OF FIGURES (Continued)

Figure | Page

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF APPENDIX TABLES

DEDICATION

For my Parents, **Dorina** and **Constantin**,

In memory of my Grandparents, **Victoria**, **Dumitru**, and **Sofia**,

For **Daciana**.

For their care, dignity, and understanding.

METHODS FOR COST-SENSITIVE LEARNING

CHAPTER 1

INTRODUCTION

1.1 Motivation

Each of us — throughout our lives — faces continually the necessity of making decisions, and our existence is governed by the consequences of these decisions. At the moment we make the decision, very often we are aware of the possible outcomes and guide our decision making process to maximize the potential benefits and minimize the losses. For example, we choose to have children vaccinated against measles, because we are willing to pay the cost of the vaccine instead of having the children at risk of getting the disease. We also consider that the lives of the children are safer by giving them the vaccine, even though it is known that there exists a very small (but non-zero) risk that the vaccine will cause permanent brain damage. Should we give the vaccine? Why? What are the risks that we take by giving the vaccine? What are the risks for not giving it? What are the losses that we might experience? These are the kinds of questions that we would like to answer as accurately as possible. Similar situations and similar questions occur very often throughout our daily lives and we would like to be able to make the best rational decisions.

Because of the large amounts of information available for most of the problems we face and because the complexity of many of the concepts that need to be handled, computational models have proven in many cases to make better and more rational decisions than humans. Therefore these models deserve our attention.

The goal of this dissertation is to present different algorithmic approaches for constructing and evaluating systems that learn from past experience to make the decisions and predictions that not only minimize the expected number or proportion of mistakes, but minimize the total costs associated with those decisions.

1.2 Machine Learning, Supervised Learning and Cost-Sensitive Learning

Machine learning is the research area that seeks to build systems that improve their performance by analyzing data organized in datasets or collected directly from the environment. During the last few years, the scientific world has been witnessing significant practical achievements of learning systems. Several mature learning algorithms have been developed and are available for practical applications.

These methods include decision rules [123] and decision trees [31, 149], neural networks [156, 189], genetic algorithms [21], Bayesian classification and statistical methods [34, 3], instance-based learning [7], multistrategy learning [124], ensemble learning [54, 73, 26], graph transformer networks [45], and support vector machines [182, 44, 14].

In practice, machine learning algorithms have been successfully applied to an increasing variety of problems such as natural language processing [46], handwriting and speech recognition [190, 95], document classification [115], medical data analysis and diagnosis [168], knowledge discovery in databases, process control and diagnosis [171], telephone fraud and network intrusion detection [145, 113], remotely-sensed image analysis [110], prediction of natural disasters [129], game playing [109], web caching and search [23], and email filtering [138].

In the meantime, a large number of empirical studies (e.g., [116], [53], [12], [55], [52]) have been performed. These studies show that there is no single algorithm that performs best in all domains.

Supervised learning is the subarea of machine learning that studies systems that learn to produce predictions for different inputs based on input-output pairs that were previously perceived (often the outputs are provided by a helpful teacher). When the set of possible predictions is discrete, this task is called *classification.*

The work described in this dissertation starts from the realization that the applications described above raise new challenges for fundamental research. Most of these problems require significant reformulation before learning algorithms can be applied, and in many cases, existing algorithms require ad-hoc tuning and modifications before being applied to a particular problem.

Before proceeding toward the design of any system, one needs to know what measures will be used to assess the performance of that system and what "good" or "improved" performance means. The desire is to have objective, realistic, and robust measures of performance. In the case of classification, the simplest performance measure can be defined as the proportion of errors (mistakes) that are made. Therefore, most existing algorithms have been designed to minimize the number of errors that are made. However, in almost all practical situations different kinds of prediction errors have different costs, and a more realistic performance measure of a learning system is the total loss, calculated as the sum of the costs of all errors made by the system. As a result, new learning approaches and new statistical evaluation methods are needed to address these problems.

For example, Kubat et al. [106] describe a learning approach to the problem of detecting oil spills from radar images of the sea surface. In this problem, the cost of indicating a spill when in fact there is none differs from the cost of missing a spill. Two other examples are described by Fawcett and Provost [68] and by Zhang and Tjoelker [171]. Both of these research teams describe their approach to building learning systems that detect potential occurrences of important events in

event sequences (for [68] the events are calls made from a cellular phone account, and the important events are the fraudulent calls made from that account). In both cases the cost of not predicting an important event when in fact one will occur (false negatives) is much smaller than the cost of predicting the occurrence of an important event when in fact none will occur (false positives). Another example is medical diagnosis, where the cost of diagnosing a patient having a life-threatening disease as being healthy is in many cases higher than diagnosing someone as being ill when she is in fact healthy.

The area of machine learning that addresses problems with non-uniform costs is known as *cost-sensitive learning.*

The goal of this dissertation is to address the practical challenges of cost-sensitive learning, introduce new learning algorithms that build cost-sensitive models, and present new statistical evaluation methods for assessing the learned models, under a general framework for cost-sensitive classification. The thesis will also describe the issues that need to be considered in the process of designing and evaluating classification methods for the minimization of the expected costs of the predictions and will review the major approaches to the cost-sensitive learning problem.

Within the scope of the thesis, the terms "cost-sensitive learning" and "cost-sensitive classification" will refer to the general problem of minimizing the total cost associated with the decisions. It is important to notice that there also exist other research questions (e.g. the order in which diagnostic tests should be performed, or the selection of training instances), which are addressed in the general cost-sensitive learning literature. Turney [178] presents an overview the various types of costs that can occur in classification problems.

1.3 Overview of the Thesis

This dissertation is structured as follows:

- Chapter 2 introduces the fundamental concepts of supervised learning, describes the formal setting of the cost-sensitive learning problem and reviews the main approaches to learning cost-sensitive models. It also argues for the necessity of good statistical evaluation methods and reviews the advantages and disadvantages of some of the current assessment methods.

- Chapter 3 presents a new general purpose wrapper method for learning and a set of heuristic methods that allow stratification methods to be used in multi-class problems.

- Chapter 4 presents the theory for applying conditional density estimation to output cost-sensitive hypotheses, introduces a new class of algorithms for class probability estimation, and shows how these algorithms can be used in practice for cost-sensitive learning

- Chapter 5 introduces new statistical methods for the cost-sensitive evaluation of learned models and discusses their relationship, advantages, and disadvantages over the ROC graph methods.

- Chapter 6 describes the experimental results of applying the newly developed learning methods to some real-world and synthetic tasks. The new statistical methods presented in Chapter 5 are evaluated experimentally.

- Chapter 7 reviews the contributions of this thesis and outlines some directions for future research.

CHAPTER 2

SUPERVISED LEARNING AND COST-SENSITIVE CLASSIFICATION

2.1 Overview

This chapter defines the supervised learning problem and the cost-sensitive classification problem. It also presents a framework for cost-sensitive learning that categorizes all existing research approaches. Basic notations and terms used throughout the dissertation are defined, and the basic cost-sensitive learning concepts are introduced. We will try to motivate the importance of studying cost-sensitive learning algorithms in the context of every-day real-world practical applications.

The task of cost-sensitive evaluation of classifiers is studied, and we analyze what issues need to be addressed by new test methods, beyond the only existing evaluation technique for cost-sensitive learning: the Receiver Operating Characteristic (*ROC*) graphs [25, 145].

Some preliminary research results that will be used later in the thesis are presented along with the basic general questions that are the subject of ongoing research in cost-sensitive learning.

2.2 Formal Setting of the Cost-Sensitive Classification Problem

In this dissertation we address aspects of the supervised learning problem. The input to a supervised learning algorithm is a set of labeled examples $\langle \mathbf{x}_i, y_i \rangle$, called *training data* (or *instance space*), where $\mathbf{x}_i$ is a vector of continuous or discrete val-

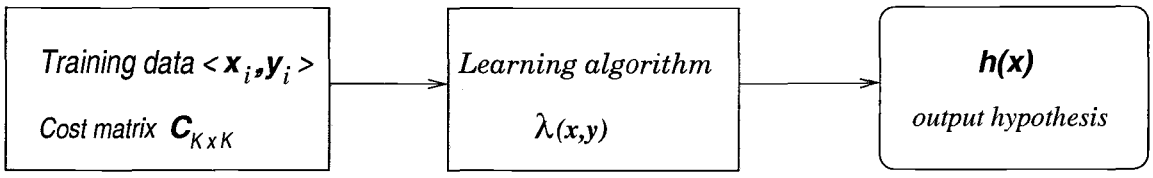


FIGURE 2.1: General framework of Supervised Learning.

ues called *attributes* and y_i is the *label* of $\mathbf{x}_i$. The training data is assumed to be generated according to an unknown probability distribution $P(\mathbf{x}, y)$. The labels can be elements of a discrete set of classes $\{1, 2, \ldots, K\}$ in the case of *classification* (or *concept learning*), or elements drawn from a continuous subset of a continuous set (e.g. a continuous subset of the reals) in the case of *regression.* The attributes are of two possible types, *numeric* and *symbolic.* A numeric attribute takes values from a subset of some ordered set of values (e.g. the set of integers). When this subset is continuous, the attribute is called a *continuous* attribute (e.g. a real valued attribute). A symbolic attribute takes values from a finite, discrete set on which no ordering relation is assumed.

Assuming that all the examples that are presented to the learning algorithm are drawn from the same (unknown) distribution *supervised learning* means finding a compact mapping h (i.e., a mathematical model) that will be able to label correctly a high proportion of unlabeled examples drawn from the same distribution. Therefore, the learning algorithm describes a procedure for building or selecting a model and tuning the model's internal parameters, such that the model will implement a function $h(x)$ that best approximates f for any examples drawn according to the same probability distribution P. We will call h a *hypothesis.*

The goal of the learning algorithm is to minimize the total expected *loss* ϵ of the hypothesis:

$$\epsilon(h) = \int P(\mathbf{x}, y)c(h(\mathbf{x}), y)d\mathbf{x}dy, \tag{2.1}$$

where $c(h(\mathbf{x}), y)$ is the cost function describing the loss incurred by h on an instance $\langle \mathbf{x}, y \rangle$.

This dissertation focuses on classification tasks and assumes that the cost function, c is static and can be represented as a K-by-K matrix C called *the cost matrix*, where K is the number of possible classes (class labels).

We will denote the possible class labels with positive integers from 1 to K, with $Y = \{1, 2, \ldots, K\}$ the set of possible class labels, and with m the number of training instances.

The Cost Matrix

The algorithms described in this thesis assume that, for a K-class problem, a K-by-K cost matrix C is available at learning time. The contents of $C(i, j)$ specify the cost incurred when an example is predicted to be in class i when in fact it belongs to class j. None of the values in C changes during the learning or the decision making process, therefore C is stationary.

In general, the cost values of a cost matrix are represented in terms of a fixed measuring unit (usually, dollar values) associated with the corresponding decisions. The learning task is not altered if all the cost values are scaled by a constant factor. The individual cost values can represent either losses or benefits, or both, and a careful analysis is needed prior to designing the matrix so that all potential costs that are incurred by a decision are captured. Losses are represented by positive values, whereas benefits are represented by negative values. By discretizing equation 2.1 and using C as the cost function, the optimal hypothesis will need to minimize

$$\epsilon(h) = \sum_{\mathbf{x},y} P(\mathbf{x}, y)C(h(\mathbf{x}), y), \tag{2.2}$$

In other words, the general objective of a classification system is to minimize the total cost of its hypothesis, computed as the sum of the costs of the individual decisions.

To illustrate the properties of a cost matrix representation of a cost function, let us consider the 2×2 cost matrix shown in Table 2.1. In this case the cost matrix represents the (approximate) costs for the decisions associated with the diagnosis of fraudulent calling card calls. The "Fraud" label indicates that a particular call is fraudulent, whereas "No-Fraud" means the call is legal. The row labels, "Fraud" and "No-Fraud" are the labels that indicate the decision of the classifier. The action associated with the "Fraud" decision is to immediately close the account involved in the call. No action is taken as a result of a "No-Fraud" decision.

TABLE 2.1: Example of a cost matrix for a calling card fraud detection problem.

	Correct Class	
Predicted Class	Fraud	No-Fraud
Fraud	−1.5	10.0
No-Fraud	100.0	0.0

In this example the cost matrix indicates that if a fraudulent call is labeled by the system as legal, there will be an average loss of about 100.0 for the provider. If

a call is mislabeled as fraudulent (and the account is mistakenly terminated), the dissatisfaction of the customer is evaluated at an average loss of 10.0. There is also a cost for correctly labeling a call as fraudulent which is represented by the cost of canceling the existing account and creating a new one for the customer. Although the actual costs of canceling an account are the same no matter whether the call was a fraud or not, it would be erroneous to construct a cost matrix that has similar values in the first row. This is true because the cost function needs to take into consideration not only actual costs but also *potential (or future) losses* (e.g. losing a customer that was annoyed by the action taken by the provider) and *potential benefits* (in the case of a good action that also has some costs associated with it, e.g. protecting the customer from fraud) as it was also pointed out by Elkan [66].

TABLE 2.2: Transformed cost matrix for the calling card fraud detection problem.

	Correct Class	
Predicted Class	Fraud	No-Fraud
Fraud	0.0	10.0
No-Fraud	101.5	0.0

The Optimal Prediction

If the probabilities for each class given an example $\mathbf{x}$, $P(y_i|\mathbf{x})$ are available, x should be labeled with y_{opt}, the class that minimizes the conditional risk of the labeling decision [63]:

$$y_{opt} = \underset{y \in Y}{\operatorname{argmin}} R(y|\mathbf{x}) = \underset{y \in Y}{\operatorname{argmin}} \sum_{j=1}^{K} P(y_j|\mathbf{x}) C(y, y_j). \tag{2.3}$$

Let h be a classifier. Let $P_h(i,j)$ be the probability that an example generated at random belongs to class j and is classified by h as being in class i. Let C be a cost matrix. Then, according to Equation 2.2, the expected loss of h on C is

$$L(h) = \sum_{i=1}^{K} \sum_{j=1}^{K} P_h(i,j) C(i,j). \tag{2.4}$$

Note that $P_h(i,j) = P_h(i|j)P(j)$, where $P(j)$ is the probability that an example belongs to class j, and $P_h(i|j)$ is the probability that h will classify an example from class j as class i.

Definition 2.2.1 *Let h_1 and h_2 be any two classifiers. Let C_1 and C_2 be two cost matrices corresponding to loss functions L_1 and L_2.*

The two cost matrices C_1 and C_2 are equivalent ($C_1 \equiv C_2$) iff, for any two classifiers h_1 and h_2, $L_1(h_1) > L_1(h_2)$ iff $L_2(h_1) > L_2(h_2)$, and $L_1(h_1) = L_1(h_2)$ iff $L_2(h_1) = L_2(h_2)$.

Theorem 2.2.1 *Let C_1 be an arbitrary cost matrix. If $C_2 = C_1 + \Delta$, where Δ is a matrix of the form*

$$\Delta = \begin{bmatrix} \delta_1 & \delta_2 & \dots & \delta_k \\ \delta_1 & \delta_2 & \dots & \delta_k \\ \cdot & \cdot & \cdot & \\ \delta_1 & \delta_2 & \dots & \delta_k \end{bmatrix} \tag{2.5}$$

Then $C_1 \equiv C_2$.

Proof: Let C_1 be an arbitrary cost matrix, and $C_2 = C_1 + \Delta$. Let h_1 and h_2 be two arbitrary classifiers.

We will prove that $L_1(h_1) - L_1(h_2) = L_2(h_1) - L_2(h_2)$ which implies $C_1 \equiv C_2$.

Let $P_1(i,j)$ be the probability that classifier h_1 classifies examples from class j as belonging to class i. Let $P_2(i,j)$ be defined analogously. According to Equation 2.2

$$\begin{aligned} L_1(h_1) &= \sum_{i=1}^{K}\sum_{j=1}^{K} P_1(i,j)C_1(i,j) = \sum_{i=1}^{K}\sum_{j=1}^{K} P_1(i|j)P(j)C_1(i,j) \\ &= \sum_{j=1}^{K} P(j)\sum_{i=1}^{K} P_1(i|j)C_1(i,j) \end{aligned} \tag{2.6}$$

and, similarly

$$\begin{aligned} L_1(h_2) &= \sum_{i=1}^{K}\sum_{j=1}^{K} P_2(i,j)C_1(i,j) = \sum_{i=1}^{K}\sum_{j=1}^{K} P_2(i|j)P(j)C_1(i,j) \\ &= \sum_{j=1}^{K} P(j)\sum_{i=1}^{K} P_2(i|j)C_1(i,j). \end{aligned} \tag{2.7}$$

If we subtract the two, we obtain

$$L_1(h_1) - L_1(h_2) = \sum_{i=1}^{K}\sum_{j=1}^{K} C_1(i,j)[P_1(i,j) - P_2(i,j)]. \tag{2.8}$$

Equivalently,

$$\begin{aligned} L_2(h_1) - L_2(h_2) &= \sum_{i=1}^{K}\sum_{j=1}^{K} C_2(i,j)[P_1(i,j) - P_2(i,j)] \\ &= \sum_{i=1}^{K}\sum_{j=1}^{K} [C_1(i,j) + \delta_j][P_1(i,j) - P_2(i,j)] \\ &= \sum_{i=1}^{K}\sum_{j=1}^{K} C_1(i,j)[P_1(i,j) - P_2(i,j)] + \sum_{i=1}^{K}\sum_{j=1}^{K} \delta_j[P_1(i,j) - P_2(i,j)] \end{aligned} \tag{2.9}$$

$$
\begin{aligned}
L_2(h_1) - L_2(h_2) &= L_1(h_1) - L_1(h_2) + \sum_{j=1}^{K} \delta_j \sum_{i=1}^{K} [P_1(i,j) - P_2(i,j)] \\
&= L_1(h_1) - L_1(h_2) + \sum_{j=1}^{K} \delta_j \sum_{i=1}^{K} [P_1(i|j)P(j) - P_2(i|j)P(j)] \\
&= L_1(h_1) - L_1(h_2) + \sum_{j=1}^{K} \delta_j P(j) \sum_{i=1}^{K} [P_1(i|j) - P_2(i|j)] \\
&= L_1(h_1) - L_1(h_2) + \sum_{j=1}^{K} \delta_j P(j) [\sum_{i=1}^{K} P_1(i|j) - \sum_{i=1}^{K} P_2(i|j)].
\end{aligned}
\tag{2.10}
$$

P_1 and P_2 are probability distributions; therefore we know that $\sum_i P_1(i|j) = 1$ and $\sum_i P_2(i|j) = 1$, and hence

$$
L_2(h_1) - L_2(h_2) = L_1(h_1) - L_1(h_2) + \sum_{j=1}^{K} \delta_j P(j)(1-1) = L_1(h_1) - L_1(h_2). \tag{2.11}
$$

So the difference in the loss of two classifiers is unaffected by adding Δ to C_1 and $C_1 \equiv C_2$. ■

The next corollary follows from this result.

Corollary 2.2.2 *A cost matrix can always be transformed into an equivalent matrix with zero values on the diagonal, or into an equivalent matrix with non-negative values.*

Proof: Let C_1 be an arbitrary cost matrix. Let Δ be a matrix of the form described by Equation 2.5.

By setting $\delta_j = -C_1(j,j)$, $1 < j < K$, the matrix C_2, with $C_2(i,j) = C_1(i,j) + \Delta$, will be equivalent to C_1 and will have the diagonal values $C_2(i,i) = 0$.

By setting $\delta_j = \min_{1 \le i \le K} C_1(i,j)$, the matrix C_3, with $C_3(i,j) = C_1(i,j) + \Delta$, will be equivalent to C_1 and will have the all cost values $C_3(i,j) \geq 0$. ■

These results show that all that matters is the relative size of $C(i,j)$ separately in each column j. So, when we generate loss matrices at random, we could generate them so that some element in each column as 0, and the others were positive. That element is not necessarily the diagonal element.

As a result, for example, the matrix in Table 2.1 can be transformed into the equivalent matrix from Table 2.2 with zero costs on the diagonal.

Also, without loss of generality, we can assume that the cost matrices have only non-negative values.

Non-stationary costs

As described above, the cost matrix assumes a uniform cost over examples from the same class that are misclassified into the same class. A more general representation of the cost function would be a cost matrix that is a function of the individual instances like the one shown in Table 2.3. This type of cost function representation would be more appropriate in applications where data points have a spatial component and f_{ij} are functions of distance measures (e.g., Burl et al. volcanoes on Venus detection task [37]).

TABLE 2.3: Example of a cost matrix which is a function of the individual instances.

		Correct Class
Predicted Class	Fraud	No-Fraud
Fraud	$-1.5 f_{00}(x)$	$10.0 f_{01}(x)$
No-Fraud	$100.0 f_{10}(x)$	$0.0 f_{11}(x)$

Another generalization of the cost function representation can be obtained by allowing it to have a temporal component [69]. For example, in Table 2.3, the f_{ij} can be functions of the time the instances were collected. This would be useful and more appropriate in problems in which the costs of the decisions change over time. For example, in the case of cellular phone fraud (e.g. cellular phone cloning), the cost of not detecting the fraud (while charges are made) increases with time, as the total amount of time the account is used fraudulently is growing and more losses result for the carrier.

In this dissertation, we will assume that none of the values in the cost matrix changes during the learning or the decision making process; therefore C is static. We will also assume that the cost matrices that are being discussed have only non-negative values.

2.3 The Three Classes of Strategies for Cost-Sensitive Learning

Conceptually, given the general supervised learning framework shown in Figure 2.1, there are three major types of strategies for cost-sensitive learning. Each of these strategies is implemented by manipulating, replacing, or altering one of the three main components of the framework: the training data, the learning algorithm, and the output of the learned model.

2.3.1 Cost-Sensitive Learning by Manipulating the Training Data

The most common practical approach to cost-sensitive classification is to modify the class distribution of the training data in order to make the learning algorithm output a hypothesis that minimizes the costs of the decisions for future examples.

For two-class problems, the simplest and most common way to do this is to present the learning algorithm with a training set in which the proportions of examples in

the two classes are changed according to the ratio of the cost values. For example, in the case of the cost matrix from Table 2.2, this approach would be implemented by training a classifier on data in which examples from the positive ("Fraud") class are 10.15(= 101.5/10.0) more numerous than the rest of the examples. This procedure is called *stratification* (or *rebalancing*), and it is usually implemented by undersampling the examples from the less expensive class, or by oversampling the examples from the more expensive class [94].

The advantage of stratification is the simplicity that makes it easy to implement for any two-class problem. However, this advantage comes at the expense of two major drawbacks: (1) in its original formulation, stratification can address only two-class problems, and (2) stratification damages the original distribution of the data by not taking into consideration the internal mechanisms used by learning algorithm to construct the hypothesis.

Chapter 3 describes new stratification procedures that generalize to multi-class problems and that also addresses the class imbalance problem (i.e., when one or more classes are underrepresented).

Another method that uses training data manipulation to learn a cost-sensitive classifier is Domingos' MetaCost [59]. MetaCost is an algorithm that employs the learned class probability estimates for the training instances and plugs them into Equation 2.3 to relabel optimally the input data. Then, it trains an arbitrary 0/1-loss classification algorithm on the relabeled data to output the cost-sensitive hypothesis. A description of MetaCost is given in Appendix B.

2.3.2 Cost-Sensitive Learning by Modifying the Learning Algorithm

Another approach to the cost-sensitive classification problem is to change the internal mechanisms of the algorithm that compute the output hypothesis such that the

algorithm will make use of the cost function (as an input parameter) to build the classifier.

There are three major approaches in machine learning algorithms that build decision boundaries to classify the data. One group of methods use heuristics that are correlated with minimizing the cost (usually, in the past, the 0/1 loss). Most algorithms for learning decision trees, decision lists, and classification rules use a mutual information heuristic. For example, each test that splits the data in a node of a decision tree is constructed based on the information gain (an entropy measure) of the distributions that result after performing the split. The second class of methods define a cost function that includes the expected loss on the training data plus a term that penalizes for overfitting. For example, neural networks are typically trained to minimize the squared loss plus a weight decay term. The third group of methods represent algorithms that combine the outputs of multiple classifiers to improve the accuracy of classification. These methods are known as ensemble methods (or committees) [51] and experimentally, it has been shown that in many applications they produce the most accurate classifications [150, 30, 53].

In the case of 0/1 loss, heuristic algorithms, and sometimes ensembles, are typically more efficient than loss-minimizing algorithms. However, it is harder to incorporate cost functions into these groups of algorithms.

Drummond and Holte [61] show that, in the case of decision trees, cost-insensitive splitting criteria can perform as well as or even better than splitting heuristics that are designed to be cost-sensitive. Their experiments also suggest that performing cost-sensitive pruning (e.g. [24]) on the cost-insensitively grown trees might reduce the cost of the decisions.

Fan and Stolfo [67] introduce a two-class cost-sensitive version of adaptive boosting, AdaCost, and show that, on some datasets, it can reduce the cost over the origi-

nal, cost-insensitive algorithm. Margineantu [118] presents two multiclass versions of cost-sensitive Bagging [27] which improve the cost on some datasets. However, the multi-class cost-sensitive version of boosting that is presented in [118] outperforms the original AdaBoost only in a few cases.

Kukar and Kononenko [108] perform a comparative study of different approaches to training cost-sensitive neural networks. Their methods include probability estimation and data manipulation techniques. Overall they show that all the algorithms under consideration reduce the costs of misclassification of cost-insensitive neural networks. However, the networks that were trained using a backpropagation version that used a cost-sensitive weight-update function performed the best.

2.3.3 Cost-Sensitive Learning by Manipulating the Outputs

To output a class label for a particular instance $\mathbf{x}$, many classification algorithms compute internally either a probability estimate of each class $\hat{P}(y|\mathbf{x}), \forall y \in Y$, or a real-valued ranking score $\hat{S}(y|\mathbf{x}), \forall y \in Y$ of the possible labels for that instance. These values can always be computed using the learned hypothesis h. For example, in the case of decision trees, the class probability distribution of an example can be estimated by the class distribution of the examples in the leaf that is reached by that example. In the case of neural networks, class ranking is usually done based on the activation values of the output units. If the learned model does not explicitly compute these values, most classifiers can be modified to output some value that reflects the internal class probability estimate. For example, Platt [141] describes how class probability estimates can be computed by training support vector machines, and Friedman et al. [76] derives the class probability estimates for Adaptive Boosting (or AdaBoost), a very accurate ensemble learning method [75].

If class probability estimates are available for an instance $\mathbf{x}$, according to Equation 2.3, its optimal label predicted y_{opt} should be computed by minimizing the estimated conditional risk

$$y_{opt} = \underset{y \in Y}{\operatorname{argmin}} \hat{R}(y|\mathbf{x}) = \underset{y \in Y}{\operatorname{argmin}} \sum_{j=1}^{K} \hat{P}(j|\mathbf{x}) C(y, j), \tag{2.12}$$

Chapter 4 introduces algorithms for estimating the underlying class probabilities $\hat{P}(y|\mathbf{x})$ of the data. These methods are compared against some of the best current approaches to computing class probability estimates.

2.3.4 Cost-Sensitive Evaluation of Classifiers

Without good statistical methods, it is difficult to tell whether any cost-sensitive methods are better than existing methods, and it is also difficult to give an estimate of the cost of a given cost-sensitive method for a particular application. Chapter 5 describes two new paired statistical tests that are the first usable methods for multi-class cost-sensitive evaluation of learned models when the cost matrix is known. In this section, we discuss the relation between these methods and cost-sensitive evaluation methods that are currently used by researchers. We will also study their advantages and disadvantages in comparison with other methods.

2.3.4.1 ROC graphs

A popular method for the cost-sensitive evaluation of classifiers is the Receiver Operating Characteristic, or *ROC* graph [84, 145]. We assume that the classification algorithm to be evaluated can output some instance class rankings. By changing the thresholds for making the decisions (based on the ranking), the learned classifier will output the whole spectrum of hypotheses that are computable by that classifier. For

a two-class problem, this spectrum ranges from predicting always class 0 (e.g. "No Fraud", or negative examples) to predicting always class 1 (e.g., "Fraud", or positive examples). The *ROC* graph of an algorithm is the curve (in the [0,1] $\times$ [0,1] space) formed by the points whose coordinates are the False Positive Rate (usually on the x axis) and the True Positive Rate (on the y axis) for different decision threshold settings. Let us denote with tp (true positives) the number of positives correctly classified, with tn (true negatives) the number of negatives correctly classified, with fp (false positives) the number of negative examples that were classified as positives, and with fn (false negatives) the number of positives classified as negatives. The True Positive Rate ($TPrate$) is defined as the ratio of the positives correctly classified to the total number of positives:

$$TPrate = \frac{\#\ of\ positives\ correctly\ classified}{total\ \#\ of\ positives} = \frac{tp}{tp + fn}. \tag{2.13}$$

Similarly, the False Positive Rate ($FPrate$) is computed as the ratio of the number of negatives incorrectly classified to the total number of negatives:

$$FPrate = \frac{\#\ of\ negatives\ incorrectly\ classified}{total\ \#\ of\ negatives} = \frac{fp}{fp + tn}. \tag{2.14}$$

The *ROC* graph shows the different cost tradeoffs available for a given algorithm for different decision threshold values. Figure 2.2 shows an example of an *ROC* curve. The origin of the graph corresponds to the classifier that always predicts the negative class. The (1,1) point of the graph corresponds to the classifier that always predicts the positive class. The best possible performance would occur at the point (1,0), where all examples would be classified correctly. All classifiers above the (0,0) - (1,1) diagonal have predictions that are better than random, while the ones below have an accuracy worse than random.

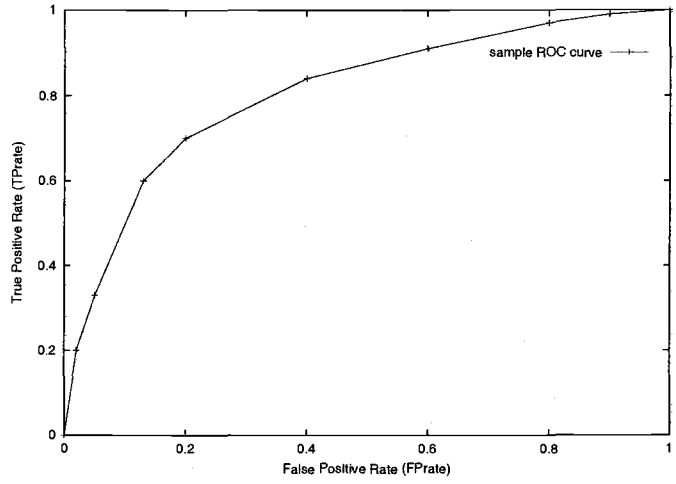


FIGURE 2.2: Example of an *ROC* curve.

The *ROC* method is designed for two-class problems, and it is a powerful method for visualizing the behavior of classification algorithms without regard to class distribution or cost matrix C.

ROC graphs are a robust method for evaluating classifiers, and there have been several attempts at inferring metrics from the *ROC* graph that would evaluate the performance of machine learning algorithms including the area under the *ROC* curve (*AUC*, [25]), and the *ROC* convex hull method [145]. Drummond and Holte [60] present a dual representation of the ROC curves called *cost curves*. This representation makes the task of comparing two (two-class) classifiers more easier. Fawcett and Provost [69] introduce a variant of the *ROC* curve that is more suitable for the evaluation of activity monitoring tasks, and tasks with a temporal component: the Activity Monitoring Operating Characteristic (*AMOC*). One disadvantage of these methods is that they are applicable only to two-class problems and their generalization to multi-class tasks is non-trivial. Hand and Till [83] show how the multi-class equivalent of the AUC can be computed using the Wilcoxon statistic.

However, probably the major disadvantage of the *ROC*-based methods (and their multi-class generalizations) is that, when used to compare two algorithms, they do not address the question of "what are the parameter settings that allow one of the algorithms to achieve the minimum cost and what is that cost?". Instead they just answer the question whether "there exists a setting for the parameters that allows one of the algorithms to win". Finding the values of the optimal parameter settings is sometimes a difficult problem, especially in the case of multi-class problems.

2.3.4.2 Ranking, ROCs and AUC

All the previously published cost-sensitive research regards the ranking and probability estimation problems as being equivalent from the cost-sensitive learning perspective. They are not. In fact the accurate ranking problem is a relaxed version of the accurate probability estimates task. If a learned model does a perfect ranking of the instances but fails to estimate accurately their class probabilities, it is hard to output the optimal decision, especially for multi-class tasks.

Let us consider the example from Table 2.4. We have two classes 0 and 1, ten instances $x_i, i = 1 \ldots 10$, and their actual class labels (in the second column). Columns 3 and 4 describe two class probability estimators γ_1 and γ_2 and their computed class probabilities $P_{\gamma_j}(1|x_i)$ for the ten examples. The examples are sorted in ascending order by the value of $P_{\gamma_j}(1|x_i)$.

The *ROC* curves corresponding to γ_1 and γ_2 are shown in Figure 2.3. They are identical because the quality of the ranking of the two learned models is the same.

Suppose the task is to minimize the total cost of the predictions for the cost matrix described in Table 2.5. The optimal classifier would set the decision threshold between x_1 and x_2 so that only x_1 is predicted to be in class 0 and all other points are predicted to be in class 1. This threshold corresponds to an average cost of

TABLE 2.4: Probability estimates γ_1 and γ_2 for a set of ten instances.

Instance	Actual class	$P_{\gamma_1}(1\|x_i)$	$P_{\gamma_2}(1\|x_i)$
x_1	0	0.01	0.35
x_2	1	0.07	0.40
x_3	0	0.30	0.65
x_4	0	0.45	0.70
x_5	0	0.60	0.72
x_6	1	0.70	0.75
x_7	1	0.75	0.77
x_8	0	0.90	0.79
x_9	1	0.92	0.8
x_{10}	1	0.95	0.85

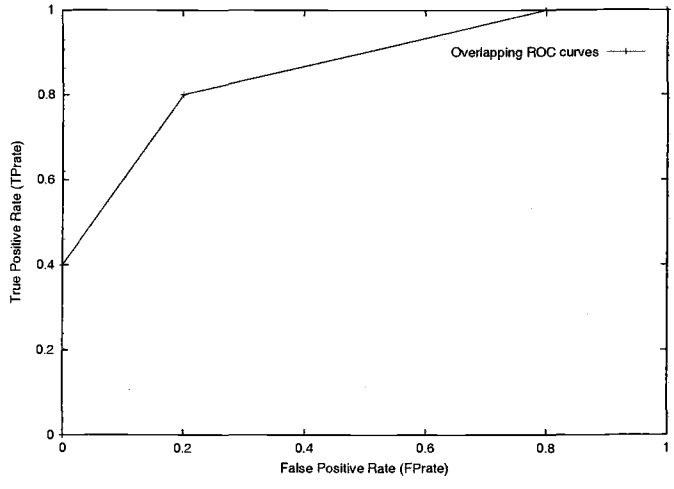


FIGURE 2.3: The *ROC* curves for γ_1 and γ_2 are completely overlapping.

4/10=0.4. If equation 2.12 is used to compute class boundaries, however, the cost incurred by γ_1 is 14 (average cost is 1.4) and the decision boundary is set between x_2 and x_3. For γ_2 the cost is 5 (an average cost of 0.5) and the computed classifier always outputs class 1.

In the case of the 0/1 loss, the error rate (computed using equation 2.12) of γ_1 is 0.3 (it would misclassify x_2, x_5 and x_8) while the error of γ_2 is 0.5 (x_2, x_3, x_4, x_5 and x_8 would be the misclassified instances). The optimal classifier, however, would set the decision boundary between x_5 and x_6 (an error rate of 0.2).

TABLE 2.5: Cost matrix for the two-class problem in Table 2.4.

	Correct Class	
Predicted Class	0	1
0	0.0	10.0
1	1.0	0.0

These are just two situations in which either one or the other of the classifiers was the winner, although their *ROC* curves are identical. However, in none of the cases could the optimal decision be computed using equation 2.12.

In most practical applications, the ultimate purpose is to minimize the cost as defined in equation 2.2. For a two-class problem, this can be translated into the Neyman-Pearson criterion [65] that states that the purpose is to find the point with the highest possible $TPrate$ that does not exceed the maximum FPrate allowed.

As we can observe from the example above, the *ROC* methods alone are not sufficient for the selection of the best classifier.

The extra step needed for the *ROC* methods is a search along the *ROC* graph for the optimal operating point. Provost and Fawcett [146] also point out this problem and provide a method, the *ROC convex hull* (ROCCH), that uses decision analysis to build the best classifier (for any parameter settings) from the models available. Their method, as well as a search along the curve that would find the best operating point, are efficient only in the case of a two-class problem. For an arbitrary number of classes K [167], the generalized versions of *ROC* methods would have to run in a new (hyper)space, which is a K^2-dimensional space to which these algorithms do not scale up.

To rectify these problems, chapter 5 presents two statistical methods for the cost-sensitive setting. Chapter 6 assesses these methods experimentally.

2.4 Summary

This chapter has defined the main concepts and notations for cost-sensitive learning. The research problems that need to be addressed, were identified, and previous approaches were reviewed. A categorization of the cost-sensitive learning was introduced, based on the framework of supervised learning: there are three main strategies of making supervised learning cost-sensitive. We have defined an equivalence relation for cost matrices and have shown how a cost matrix can be transformed into an equivalent one. We have described the *ROC* graph, a method for evaluating and visualizing the quality of the rankings for two-class problems. However, *ROC* methods do not show how to set classification thresholds to minimize misclassification costs and are not applicable to multi-class problems. We have shown that if class probability estimates are not accurate, equation 2.3 cannot be used to classify examples,

even if the estimates represent very good rankings. This motivates the need for new statistical methods for cost-sensitive evaluation.

CHAPTER 3

COST-SENSITIVE LEARNING BY TRAINING DATA MANIPULATION

3.1 Overview

In supervised learning, we assume that the data sample is drawn according to a probability distribution $P(x, y)$ and hence, the class labels will have probabilities $P(y)$. An important method for making a classifier cost sensitive is to alter the proportion of training examples or their labels to reflect a different distribution $P'(y)$. If the learning algorithm is sensitive to $P(y)$, this will cause it to shift its decision boundaries. For example, if the cost matrix for a problem is the one shown in Table 3.1, one might decide to grow a decision tree model by presenting the decision tree learning algorithm a sample that contains ten copies of each instance labeled with Class 1 in the original data. The reasoning behind this heuristic is that Class 1 is ten times more expensive to misclassify than Class 0, and the learned model will be able to minimize the overall cost if the training is preformed on data in which examples from Class 1 are more abundant.

This chapter studies methods for changing the class distribution of the original training data such that the learning algorithm will construct a cost-sensitive classifier.

In practice it is usually the case that the original class distribution is distorted to have each class represented in proportion to its "importance" to the learning algorithm at the time of training. In the case of two-class problems, the level of "importance" is computed using a function of the value of the misclassification cost of the class and the original priors (class distribution). Some machine learning software

TABLE 3.1: Example of a two-class cost matrix for which a good heuristic to learn a cost-sensitive model would be to change the original class distribution by sampling each instance from Class 1 ten times and leave the number of examples from Class 0 unchanged. c is an arbitrary positive cost value.

	Correct Class	
Predicted Class	Fraud	No-Fraud
Fraud	0.0	$10c$
No-Fraud	c	0.0

packages allow the user to specify the "importance" in the form of weights that are associated either to individual examples or to class labels. This is the reason why the learning techniques that use modified class priors are also known as *weighting* methods and the priors are sometimes called *weights*.

Once the weights are computed, if the base learning algorithm λ does not provide an explicit way of incorporating them (e.g., the C4.5 decision tree learning package provides the *Weights[]* variable), the simplest implementation of the weighting mechanism is to present the algorithm, at training time, each example in proportion to its weight.

Weighting is a technique that can be applied to any classification algorithm. It has the advantage of providing a powerful means for changing the learned classifier, while the user is not required to know all the implementation details and internal mechanisms of the learning algorithm.

The general framework of learning cost-sensitive models by weighting is shown in Figure 3.1, and the pseudo-code for it is given in Table 3.2. The algorithm assumes that the cost matrix and the base 0/1-loss learning algorithms are available.

Weighting exploits the sensitivity of learning algorithms to the input data distribution.

TABLE 3.2: CS-DATAMANIP, a general procedure for learning cost-sensitive models via training data manipulation.

CS-DATAMANIP

Input: Training data $D = \langle \mathbf{x}_i, y_i \rangle$, cost matrix C, 0/1-loss learning algorithm λ

(K is the number of classes)

// Initialize the weights.

for i **from** 1 **to** K, **let** $w_i = 1.0$

// Update the weights. Some procedures like the wrapper method described in Section 3.2.1

// make use of the learning procedure λ to compute the new set of weights.

w = UPDATEWEIGHTS($\mathbf{x}$, $\mathbf{y}$, C, λ)

// Use the 0/1-loss learning algorithm to learn the cost-sensitive model

$h = \lambda$ ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$)

Output: h

end CS-DATAMANIP

For a two-class task, with a generic cost matrix like the one described in Table 3.3, the problem of finding the optimal class weights is equivalent to finding the class weights for 0/1-loss learning but for some test data distribution in which the examples from Class 1 have a frequency of occurrence which is c_{10}/c_{01} higher than their frequency in the training data. If we denote the probability values that refer to the new test data distribution with P_s, then $P_s(1) = P(1)c_{10}/c_{01}$ and $P_s(0) = P(0)c_{01}/c_{10}$.

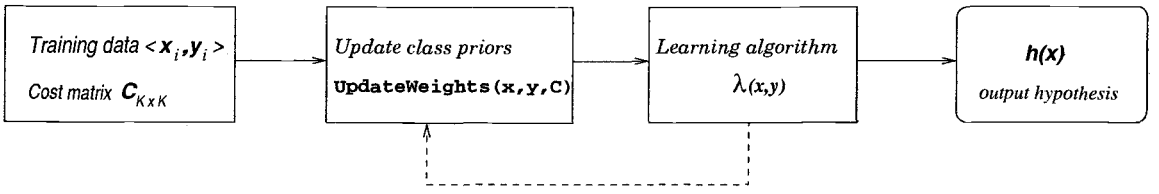


FIGURE 3.1: General framework for Cost-Sensitive Learning via Changing Class Weights. The dashed line indicates that some procedures might use the learning algorithm for computing the class weights.

TABLE 3.3: Example of a generic two-class cost matrix.

	Correct Class	
Predicted Class	0	1
0	0	c_{10}
1	c_{01}	0

This means that the examples from Class 1 should be weighted by c_{10}/c_{01}, or equivalently, the examples from Class 0 should be weighted by c_{01}/c_{10}.

For a K-class task, learning by weighting makes use of the K weight parameters, whereas the cost matrix has $K(K-1)-1$ free parameters. As a result, for $K > 2$, there is, in general, no way to exactly implement the cost matrix using the weight parameters. In other words, because for more than two classes there exists, in general, more than one non-zero misclassification cost for each class (there is a misclassification cost for each pair of classes) - the two-class weighting procedure can not be generalized for arbitrary K-class tasks. Nonetheless, it might be possible to set the weights to approximately implement the cost matrix.

Our primary goal in this chapter is to answer the question of how the class weights should be set to train multi-class learners when the misclassification costs are known. In other words, we will study different algorithms for the implementation of the UPDATEWEIGHTS procedure in Figure 3.1. One wrapper method and two types of heuristic approaches are presented and discussed.

3.2 Multi-class Weighting Methods

We present five different methods for choosing the input weight vector $\mathbf{w}$. These methods can be divided into three different categories: wrapper methods, class distribution-based methods, and cost-based methods. This section describes each of these in turn.

3.2.1 A Wrapper Method

The most expensive approach is a "wrapper" method, which treats the learning algorithm as a subroutine (i.e., this method might take the path described by the dashed line in the framework from Figure 3.1) as follows. Suppose we randomly partition our training data into a subtraining set and a validation set. When the learning algorithm is applied to the subtraining set with input weight vector $\mathbf{w}$, the cost of the resulting model can be measured using the validation set and the cost matrix C. Hence, we can use a general-purpose optimization algorithm to generate different $\mathbf{w}$'s and try to find the $\mathbf{w}$ that minimizes the cost on the validation set.

We chose Powell's method [143], which is a gradient-free, quadratically-convergent optimization algorithm. The method is described by the pseudo-code in Table 3.4. It works by performing a sequence of one-dimensional optimizations. The search directions are chosen to try to make them orthogonal, so that rapid progress is made toward the minimum.

TABLE 3.4: Algorithm that uses Powell's optimization method to compute the class weights.

POWELL-UPDATEWEIGHTS

Input: Training data $D = \langle \mathbf{x}_i, y_i \rangle$, cost matrix C, 0/1-loss learning algorithm λ,

p_V the proportion of training data used for validation

$\mathbf{w}$ the weights, initially having randomly assigned values

let $\mathbf{w}_{saved} = \mathbf{w}$; // save the initial values of the weights

for $d = 1$ **to** K

// initialize the set of directions $\mathbf{u}_d$ with the basis vectors (1 in the d^{th} position, 0 otherwise)

let $\mathbf{u}_d = (0, \ldots, 0, 1, 0, \ldots, 0)$; // where 1 is in the d^{th} position

end // for d

repeat

let $\mathbf{w}_0 = \mathbf{w}_{saved}$; //starting point

for d **from** 1 **to** K //all directions

// perform a line search

// **call** λ and **move** $\mathbf{w}_{d-1}$ along direction $\mathbf{u}_d$ to a minimum; call it $\mathbf{w}_d$

let $\alpha_d = \underset{\alpha}{\text{argmin}}\ \lambda(\mathbf{w}_{d-1} + \alpha \mathbf{u}_d)$; **let** $\mathbf{w}_d = \mathbf{w}_{d-1} + \alpha_d \mathbf{u}_d$; **let** $\mathbf{u}_d = \alpha_d \mathbf{u}_d$;

end // for d

for d **from** 1 **to** $K - 1$ **let** $\mathbf{u}_d = \mathbf{u}_{d+1}$;

let $\mathbf{u}_K = \mathbf{w}_K - \mathbf{w}_0$;

// perform a line search

// **call** λ and **move** $\mathbf{w}_K$ along $\mathbf{u}_K$ to a minimum; call it $\mathbf{w}_0$

let $\alpha_0 = \underset{\alpha}{\text{argmin}}\, \lambda(\mathbf{w}_K + \alpha \mathbf{u}_K)$; **let** $\mathbf{w}_0 = \mathbf{w}_K + \alpha_0 \mathbf{u}_K$; **let** $\mathbf{u}_0 = \alpha_0 \mathbf{u}_K$;

until convergence of $\mathbf{w}$;

Output: $\mathbf{w}$

end POWELL-UPDATEWEIGHTS

The overall procedure is the following: Powell's algorithm is executed using the sub-training set and the validation set to determine the values of the weights $\mathbf{w}$. Then, the final model (e.g. decision tree) is built using $\mathbf{w}$ on the whole training set.

3.2.2 Class Distribution-based Methods

The second group of methods is based on making some measurements of the frequencies of the different classes and different errors on the training set and then computing a good input vector $\mathbf{w}$ from this information.

Most prediction errors committed by learning algorithms tend to result from predicting items that belong to less frequent classes as being in more frequent classes. This observation suggests the following two methods.

The first method in this category is *ClassFreq*. In this method, we first compute the class frequencies in the training data, such that n_i is the number of training examples belonging to class i. Then, the input weights to the 0/1-loss classification algorithm are set so that $n_i \times w_i$ is constant for all classes $1 \leq i \leq K$. This gives higher weight to classes that are less frequent.

The second method, called *EvalCount*, is based on first learning a model to minimize the traditional 0/1-loss. This is accomplished by subdividing the training data into a sub-training set and a validation set, constructing a classifier on the sub-training set, and then measuring its costs (according to cost matrix C) on the validation set. We then set w_i to be the sum of the costs of the examples from class i that were misclassified, plus 1 (to avoid having weights with a value of zero).

As with the wrapper method, we need to set another input parameter, v, the proportion of training data used for validation.

3.2.3 Cost-based Methods

The third group of methods for computing **w** calculates them directly from the cost matrix without either invoking the learning algorithm or measuring class frequencies in the training data. We call such methods *cost-based methods.* The big advantage of these methods is that they are extremely efficient. Two cost-based methods are proposed for calculating the weights.

The first method is called *MaxCost.* Each weight is computed as the maximum of the corresponding column:

$$w_j = \max_{1 \leq i \leq k} C(i,j) \tag{3.1}$$

The intuition is that the maximum value within a column is the worst-case cost of misclassifying an example whose true class corresponds to that column.

The second method is called *AvgCost* and is similar to the one suggested by Breiman et al. [31]. Each weight is computed as the mean of the off-diagonal elements in the corresponding column:

$$w_j = \frac{\sum_{i=1, i \neq j}^{k} C_k(i,j)}{(k-1)} \tag{3.2}$$

The intuition here is that the average of the non-zero values within a column approximates that average cost of misclassifying an example whose true class corresponds to that column.

3.3 Summary

We have presented five weighting methods for multi-class tasks. The wrapper method uses the base learning algorithm as an internal procedure; therefore we expect it to

be more accurate than the other methods in certain situations. The cost-based algorithm can give poorer results especially when the misclassification costs take values from a large interval, because these methods will create very imbalanced data distributions, which may cause the learning algorithm to overfit. The distribution-based methods will create the same problem if the original distribution of the data is skewed.

Unfortunately, the wrapper approach is also the most expensive to execute, so this raises the question of whether some of the cheaper methods will work well in commonly-occuring situations.

CHAPTER 4

COST-SENSITIVE LEARNING BY CONDITIONAL DENSITY ESTIMATION

4.1 Overview

In the case of a K-class problem where the cost matrix C is known, an instance x should be labeled with the class γ that minimizes the conditional risk (or, expected loss) for that instance

$$R(\gamma|x) = \sum_{j=1}^{K} P(j|x)C(\gamma, j). \tag{4.1}$$

This equation gives us the optimal labels if the probabilities $P(j|x), \forall j = 1 \ldots K$, are accurately computed. In this chapter we will study different approaches for learning accurate class probability estimates and how these methods should be employed for cost-sensitive classification. First we will review one of the best existing methods, the bagged probability estimation trees (B-PETs). Then we will outline the potential advantages that the lazy learning framework offers for class probability learning and will introduce a new approach, the lazy option trees (LOTs) and the bagged lazy option trees (B-LOTs).

4.2 Decision Trees

One of the most studied 0/1-loss learning algorithms is the decision tree algorithm [31, 148, 149]. It produces tree models like the one shown in Figure 4.1 for classification. The decision trees consist of internal nodes and leaves. The internal nodes

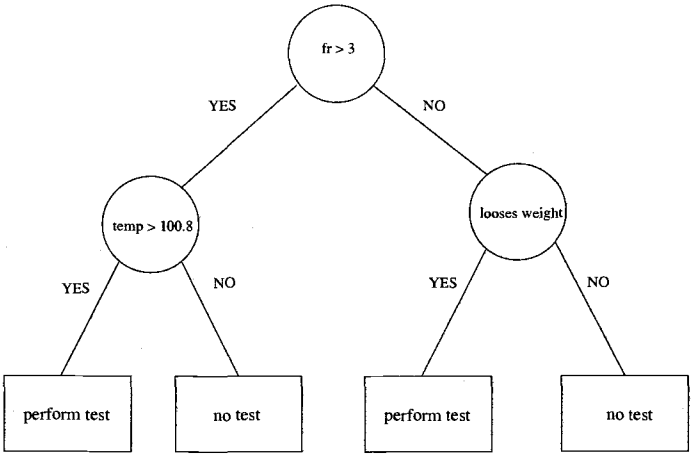


FIGURE 4.1: Example of a binary decision tree for deciding whether a patient is suspect of having AIDS, and therefore whether to perform an HIV test. Internal nodes representing tests on possible attribute values, are represented by circles (fr is the frequency of recurrent infections per month, *temp* is the body temperature). Leaf nodes indicating the decision, are represented by rectangles.

specify tests on attribute values while the leaves specify class labels. To classify an unlabeled example, the decision tree will perform the attribute test specified by the root node and follow the branch corresponding to the outcome of the test (e.g., an instance with $fr = 1$ will follow the NO branch in the tree from Figure 4.1). In the case of reaching another internal node, the instance will traverse the tree through the branches corresponding to the outcomes of the tests, all the way to a leaf node. When a leaf node is reached, the example will be assigned the class label specified by the leaf.

The standard binary decision tree induction algorithm is described in Table 4.1. In each node, all possible attribute tests are considered and assessed (on the training data) based on some evaluation function (usually a heuristic). The test with the highest score is chosen, and the training data is split using the test. The induction procedure is called recursively for each partition of the data. The splitting process

halts when any one of the following conditions is satisfied: (1) all training instances reaching the node belong to the same class, (2) all training instances have the same attribute values, (3) the number of instances is smaller than the minimum number allowed (a parameter given by the user), or (4) the assessment heuristic indicates that no further improvement of the model can be achieved. Most of the heuristic evaluation functions for choosing a test that are used in practice, make use of some measure of the purity of the data (i.e., seek tests that lead to nodes in which the number of instances in one class is much larger than the instances from other classes, ideally having only instances from a single class). CART [31] uses the Gini criterion, whereas ID3 [148] and C4.5 and C5 [149] employ a function of the entropy of the data (information gain and gain ratio). Other possible splitting criteria include the g-criterion invented by Kearns, and Mansour [98], criteria based on the χ^2 and G^2 statistics [126], and meta-criteria based on the p-value of any of the above tests [58].

When the tree model is constructed, a pruning procedure is employed in order to avoid overfitting the input data. Three categories of pruning methods are commonly used in practice for pruning trees for 0/1-loss classification. Holdout and cross-validation methods use a portion of the training data for validation, and the model is pruned to minimize an error function on the validation data. Cost-complexity pruning [31] used in CART and reduce-error pruning fall in this category. The second category includes methods that rely on an information-theoretical approach that seeks the model with minimal complexity, for example based on the minimum description length (MDL) criterion [153, 10]. The third category is represented by methods that rely on a probabilistic estimate of the error based on the counts of instances in each node. Pessimistic error pruning [149] used in the C4.5 programs belongs to this category.

TABLE 4.1: The standard binary decision tree induction algorithm.

DECISIONTREE

Input: Training data $D = \langle \mathbf{x}_i, y_i \rangle$, $MaxExamples$ a constant

if all instances in D have the same label y_l **then return** $Leaf(y_l)$

if all instances in D have the same attribute values

then $y_l = MajorityLabel(D)$; **return** $Leaf(y_l)$

if $card(D) < MaxExamples$ **then** $y_l = MajorityLabel(D)$; **return** $Leaf(y_l)$

select an attribute test T

let D_{pos} = all instances in D for which the outcome of the test T is positive

let D_{neg} = all instances in D for which the outcome of the test T is negative

// now, call the procedure recursively

let LeftBranch = DECISIONTREE(D_{pos}, $MaxExamples$);

let RightBranch = DECISIONTREE(D_{neg}, $MaxExamples$);

Output: $Node(T, LeftBranch, RightBranch)$;

end DECISIONTREE

Decision trees are popular among practitioners (to give just a few examples of their use in practical applications, see [91, 159, 147, 95, 158]) because fast induction algorithms are available, because they classify new examples quickly, and because they can output rules that are understandable by users. Typically, the probability estimates for an unseen instance that reaches a leaf l are approximated using the class counts of training instances that reach l. For example, if the rightmost leaf in Figure 4.1 is reached by 4 instances from class *perform test* and 0 instances from the *no test* class, the probability estimated using the tree, for an instance x that reaches the same leaf is $P(perform\ test|x) = 4/4 = 1.0$ and $P(no\ test|x) = 0/4 = 0.0$.

As noted by [165, 25, 147, 29, 144], the class probability estimates of the decision trees are poor. There are three major factors that cause this deficiency. First, the greedy induction mechanism that splits the data into smaller and smaller sets leads to probability estimates that are computed based on very small samples, and this leads to inaccurate estimates. Second, most of the existing decision-tree induction algorithms focus on minimizing the number misclassifications (through the purity-based heuristics) and on minimizing the size of the model (through the pruning procedure). This causes the learned models to compute class probabilities that are too extreme (i.e., close to 0.0 and 1.0), as in the example above, and therefore incorrect. The third factor is the shape of the decision tree hypotheses (piecewise linear decision boundaries). This kind of decision space assigns uniform probability values to points that are in the same region and will not differentiate between points that are closer to the boundary of the region and points that are farther from the boundary.

4.3 Probability Estimation Trees

Provost and Domingos [144] show that decision tree class probability estimates can be improved by skipping the pruning phase and smoothing the distributions by applying a Laplace correction (or Dirichlet prior) as follows

$$P(y_j|x) = \frac{N_j + \lambda_j}{N + \sum_{i=1}^{K} \lambda_i} \tag{4.2}$$

where N is the total number of training examples that reach the leaf, N_j is the number of examples from class y_j reaching the leaf, K is the number of classes, and λ_j is the prior for class y_j (assumed to be uniform $\lambda_i = 1.0\ \forall i = 1 \ldots K$ in this case, and

in all other applications in which there is no prior knowledge about the distribution of the instances). The Laplace correction [80, 39, 24] will smooth probability estimates that are too extreme because of the small size of the sample that reaches the leaf. This smoothing permits Probability Estimation Trees to reduce the effects of the second of the causes for inaccurate estimates (extreme probabilities), described in the previous section. To handle the other two sources of inaccuracy of tree-based probability estimates, Provost and Domingos apply Bagging [31]. The resulting models are called Bagged Probability Estimation Trees (or B-PETs). Bagging averages the probabilities computed by multiple models. Each of the models is trained using a bootstrap replicate [64] of the training data. The Bagging algorithm is described in detail in Appendix B.

4.4 Lazy Learning

We propose an alternative to B-PETs based on lazy learning. Lazy learning is the framework in which a model is built only when the attribute values of the instance to be classified are known. Commonly, under this framework a model (ideally, "the best model") is built for each individual unlabeled test instance. The most popular example of a lazy learning algorithm is the nearest neighbor algorithm [87, 172, 47, 187], which classifies an example based on the labels of the instances that are closest to it (according to some distance measure). The major advantage of lazy learning is that it exploits knowledge of (at least) some of the attribute values of the instance to be classified and therefore has the potential of building more accurate and more compact models focusing solely on the query instance. The disadvantage is that classification time can be significantly larger for lazy algorithms, and this can affect their utility in some practical applications.

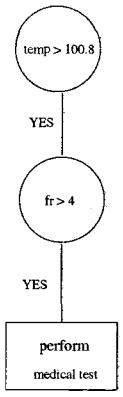


FIGURE 4.2: Sample decision "tree" built by the lazy decision tree algorithm for a specific unlabeled instance $\langle fr = 6, weight\ loss = NO, blood\ pressure = HIGH, body\ temperature = 101.5 \rangle$.

Our decision to employ the lazy learning framework is based on the intuition that lazy models can capture more accurately the specific characteristics of previously unseen instances and the neighborhood around them and therefore may be able to compute better probabilities.

4.5 Lazy Trees

In order to learn better class probability estimates, we start by employing a combination of the lazy learning idea and decision tree algorithms, an algorithm called lazy decision trees introduced by Friedman et al. [34, 77]. The lazy decision tree algorithm builds a separate decision tree for each test instance (the query point). Because only one instance has to be classified by the learned model, each internal node will have only one outgoing branch — the one that tests positive on the query point. For each internal node, the selected test will be the test that maximally decreases the entropy for the node to which the test instance would branch. The information gain is defined to be the difference between the two entropy values. Table 4.2 outlines the

generic lazy decision tree learning algorithm, and Figure 4.2 shows an example of a decision tree induced by the lazy decision tree algorithm.

TABLE 4.2: The lazy decision tree learning algorithm.

LAZYDECISIONTREE(D,$\mathbf{x}_u$,$MaxExamples$)

Input: Training data $D = \langle \mathbf{x}_i, y_i \rangle$, Unlabeled example $\mathbf{x}_u$, $MaxExamples$ a constant

if all instances in D have the same label l **Output** D

if all instances in D have the same attribute values **Output** D

if $card(D) < MaxExamples$ **Output** D // D has less than $MaxExamples$ instances

select an attribute test T

let t_u = the value of test T on $\mathbf{x}_u$

let D_n = all instances in D for which the value of the test T is t_u

LAZYDECISIONTREE(D_n, $\mathbf{x}_u$, $MaxExamples$); // call the procedure recursively

end LAZYDECISIONTREE

The most straightforward estimate of the class probabilities can be computed using the class label counts of training data D in the leaf node

$$P(y_j|x) = \frac{N_j}{N}, \tag{4.3}$$

where N is the total number of examples in D, and N_j is the number of examples from class y_j in D.

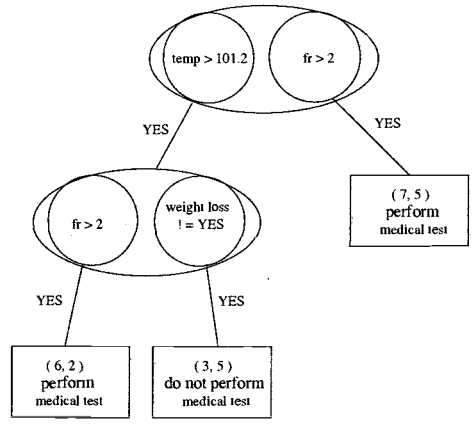


FIGURE 4.3: Sample tree built by the lazy option tree algorithm for a specific unlabeled instance $\langle fr = 6, weight\ loss = NO, blood\ pressure = HIGH, body\ temperature = 101.5\rangle$. The numbers in the leaves indicate the number of training examples (from each of the classes, respectively) that reach that leaf. Each leaf computes a probability estimate based on those numbers. These estimates are then averaged to produce the final estimates. The labels of the leaves indicate the decision in the case of 0/1 loss.

4.6 Multiple Options at Nodes: Lazy Option Trees (LOTs)

Although, by their ability to focus on individual unlabeled instances, the lazy trees attempt to avoid inaccurate estimates of points too close to the decision boundary, the small sample and the greediness of the induction method are still influencing the quality of the probabilities.

To correct this, instead of having a single test in the internal nodes, we allow multiple tests (options) in each node to grow lazy option trees (or, LOTs). We have extended Wray Buntine's [34] and Kohavi's and Kunz' [101] ideas of option decision trees for classification, into the lazy tree learning framework. The lazy option tree algorithm selects in each node the best b_t tests with the highest information gain.

A lazy option tree example that was constructed for the instance $\mathbf{x}_u = \langle fr =$

$6, weight\ loss = NO, blood\ pressure = HIGH, body\ temperature = 101.5\rangle$, is depicted in Figure 4.3. Let us denote as class 1 the *perform medical test* class and as class 2 the *do not perform medical test* class. To compute the class probability $P(1|\mathbf{x}_u)$, the algorithm will calculate the proportion of training examples from class 1 from each of the three leaves and will average the values. This corresponds to taking all the paths from the root node to the leaf node. Because the tree was built just for $\mathbf{x}_u$, all the tests on these paths will be satisfied by the attribute values of $\mathbf{x}_u$. In this case $P(1|\mathbf{x}_u) = (0.75 + 0.375 + 0.583)/3 \approx 0.5694$, and $P(2|\mathbf{x}_u) = 1.0 - P(1|\mathbf{x}_u) \approx 0.4306$.

The options represent an alternative to the voting mechanism of Bagging, for smoothing and improving the probability estimates of the tree models. The advantage of the options mechanism is that tests having an information gain almost as high as the best test will be performed, while they might never be performed in decision trees, lazy trees, or even bagged trees. This way, diversity is increased, and this might lead to better probability estimates. In addition, lazy option trees (LOTs) offer a single compact model that is comprehensible.

Nonetheless, to improve the class probability estimates of the LOTs we propose applying bagging on top of the algorithm, resulting in bagged lazy option trees (B-LOTs). Our intuition is that the different nature of the two mechanisms, options and bagging, will help improving the computed estimates. In the case of LOTs and B-LOTs, the user will have to set two additional parameters: $MaxTests$—the maximum number of tests that are allowed in a node, and g—the minimum gain proportion ($0.0 < MinG < 1.0$), a number indicating the minimum gain for a test in order to be selected. g is the proportion out of the gain of the best test achieved within that node (e.g., if the best test in a node has a gain on g, we will will discard all tests that have a gain of less than gI, within that same node).

4.7 Summary

The issue of cost-sensitive learning via class probability estimates has been addressed. We have presented a new class probability estimation approach based on the idea of lazy tree learning. We have shown that decision trees produce poor class probability estimates for three reasons: (1) their focus on minimizing the 0/1 loss (the misclassification rate) leads to extreme estimates (close to 0 and 1), (2) they generate piecewise constant estimates over large regions, and (3) fragmented data. The PETs reduce the first two causes, and the B-PETs also address the third.

We have proposed a lazy learning solution, the B-LOTs. Lazy learning focuses on a specific instance and will tailor the learned model for it. The intuition is that the options in the nodes improve ensembles over bagging.

CHAPTER 5

THE COST-SENSITIVE EVALUATION OF CLASSIFIERS

5.1 Overview

A fundamental issue in developing machine learning algorithms for applications is the assessment of their performance. Evaluation of the algorithms should be one of the first things to consider, rather than an afterthought. Evaluation is a complicated issue, especially if considered in the context of real-world applications. To evaluate a machine learning algorithm, we need to systematically assess its performance. This will help us to understand *how* a system performs, *how* to solve a problem, and *what* are the causes and effects of different patterns in the data.

Previous work that described multi-class cost-sensitive classification methods has either applied no statistical testing or has applied statistical methods that are not appropriate for the cost-sensitive setting. We need good statistical methods to tell whether new cost-sensitive methods are better than existing methods that ignore costs, and to estimate the cost of a given cost-sensitive method for an application.

Evaluation of 0/1-loss algorithms has turned out to be difficult. The most used method for predicting out-of-sample errors is 10-fold cross-validation, and for comparing misclassification rates of two learning algorithms are McNemar's test, the 10-fold cross-validated t test, and the 5x2cv test [9, 52].

To rectify these problems, this chapter presents two statistical methods for the cost-sensitive setting. The first constructs a confidence interval for the expected cost of a single classifier to answer the question "what is the performance of classifier γ?".

This is useful for estimating the costs that are likely to be incurred on a given practical task and also for determining whether the available test data set is large enough to estimate the expected cost accurately. The second method constructs a confidence interval for the expected difference in cost of two classifiers to answer the question "which classifier performs better, γ_1 or γ_2?". This is useful for comparing classifiers for a particular application, and it will also help guide research on cost-sensitive learning. In both cases, the basic idea is to separate the problem of estimating the probabilities of each cell in the confusion matrix from the problem of computing the expected cost.

Our tests are based on the idea of *bootstrap* [64], a computational method that is used for estimating the standard error of a parameter q (e.g., the mean) of an unknown distribution D, based on a random sample $\mathrm{s} = (s_1, s_2, \ldots, s_n)$ drawn from that distribution. The bootstrap works by drawing *with replacement* T samples $\mathrm{s}^1, \mathrm{s}^2, \ldots, \mathrm{s}^T$ (each consisting of n data values), from s. the value parameter of interest q is computed for each of these samples $q^1, q^2, \ldots, q^T$. The standard error of q is estimated by the sample standard deviations of the T replicates (also called *bootstrap replicates*).

We show both theoretically and experimentally that our new tests work better than applying standard z tests based on the normal distribution. The two statistical tests are designed for the cost-sensitive setting where the cost matrix C is available. We do not treat the case where C is not available, although one strategy might be to apply the methods of this chapter to several different C matrices drawn by sampling from some distribution of likely cost matrices.

5.2 Bootstrap Methods for Cost-Sensitive Classification

Cost-sensitive evaluation of learned models is made difficult by the fact that in many domains the expensive errors correspond to the rare cases. In air traffic control, for example, crashing an aircraft is a very expensive error, but (fortunately) there are very few training examples of airplane crashes, compared to the number of safe flights.

In the ordinary 0/1 loss case, we can group together all of the different misclassification errors and estimate the total fraction of misclassifications. But in the cost-sensitive case, we need to treat each different kind of misclassification error separately, because they have different costs.

The difficulty of the statistical problem created by the scarcity of expensive examples is illustrated in Figure 5.1, which plots the distribution of the total loss on 1000 separate test sets for a fixed classifier and cost matrix. For different test sets, the costs varied over a huge range because of the presence or absence of rare, expensive errors. In this case, a large test set is needed to ensure that we obtain enough data to be able to detect these rare, expensive cases.

The theoretical motivation of this distribution and why normal distribution based t or z tests fail, is simple. Given a set of n test examples, we can apply the learned classifier γ to evaluate each example. Let the cost of classifying example $(\mathbf{x}_\ell, y_\ell)$ be $c_\ell = C(\gamma(\mathbf{x}_\ell), y_\ell)$. We can compute the mean and standard deviation of the c_ℓ values and construct a confidence interval based on the t or z statistics. The central limit theorem [185] assures us that the mean of n cost values will be normally distributed with a variance of $\sigma/\sqrt{n}$, where σ is the variance of the c_ℓ values. Unfortunately if the values in the cost matrix C are very different, the variance σ will be very large, so the resulting confidence interval will be very wide. Consequently, this procedure

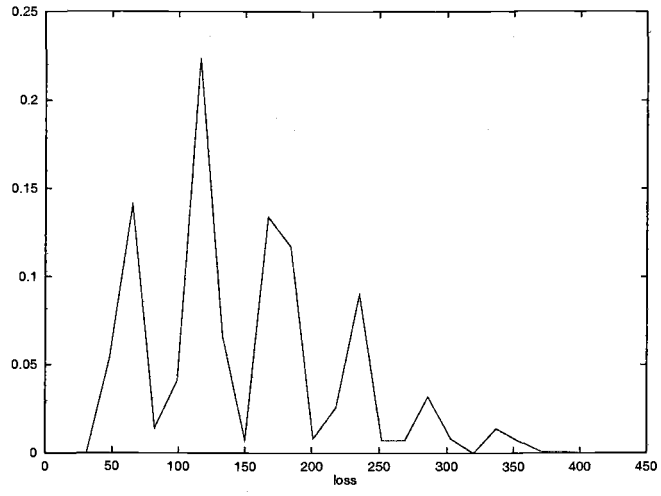


FIGURE 5.1: Example of distribution of losses for a classifier and a cost matrix over 1000 different test sets drawn from the same underlying distribution.

will not give us a very tight confidence interval.

The alternative approach is to estimate the probability on the test set, for each kind of mistake. Let $p(i, j)$ be the probability that a test example belonging to class j was predicted to be in class i. These probability values can be computed by constructing a confusion matrix and dividing each cell by n. Once these probabilities are computed, we can compute the expected cost of the classifier by multiplying each element $p(i, j)$ by the corresponding cell $C(i, j)$ and summing the results:

$$\overline{c} = \sum_{i,j} p(i, j) \cdot C(i, j). \tag{5.1}$$

Based on a small test set, there will be some uncertainty about each of the $p(i, j)$ values. If the corresponding cost $C(i, j)$ is very large, then this uncertainty will have a big impact on the estimated error. Fortunately, each probability $p(i, j)$ is a better-behaved statistic than the average of the c values. It has a variance that

depends only on its true value, and extreme values (near zero and one) have smaller variances than intermediate values. Moreover, when the test data set is small, we can use Laplace corrections (or equivalently, Dirichlet priors) to incorporate prior beliefs about the sizes of the $p(i,j)$ values [80, 39].

The approach that we take is to compute the $p(i,j)$ values, adjust them using uniform Laplace corrections, and then apply bootstrap methods to construct confidence intervals for the expected cost of the classifier. We will see that this approach gives much better results than simply computing a confidence interval based on the t or z statistics.

5.2.1 *Estimating the Expected Cost of a Classifier:* BCOST

Let γ be a classifier, and $\mathcal{D}$ be a test set of n examples. Let M be the $K \times K$ confusion matrix computed from γ and $\mathcal{D}$, where element $M(i,j)$ is the number of test set examples predicted by γ to be in class i but actually belonging to class j.

Let $p(i,j)$ be the normalized, Laplace-corrected confusion matrix defined as follows:

$$p(i,j) = \frac{M(i,j) + \lambda}{K^2\lambda + n}. \tag{5.2}$$

Here, the constant $\lambda \geq 0$ determines the strength of the Laplace correction. The $p(i,j)$ values can be viewed as a multinomial probability distribution over the K^2 combinations of predicted and correct classes.

The BCOST procedure computes a confidence interval for the expected cost of classifier γ as shown in Table 5.1. This pseudo-code generates N_b simulated confusion matrices $\tilde{M}_u$ by generating a sample of size n according to the distribution $p(i,j)$. For each simulated confusion matrix, it then computes the cost $\tilde{c}_u$ by taking a dot

product between $\tilde{M}_u$ and the cost matrix C. This gives N_b simulated costs. For a 95% confidence interval, it then outputs the first and the last of the middle 95% values after sorting them into ascending order. For example, if the number of bootstrap rounds is 1000 ($N_b = 1000$), it will sort the 1000 values and will output the 26th and 975th simulated costs. This is a bootstrap confidence interval for the mean cost after applying a Laplace correction to the original confusion matrix.

TABLE 5.1: The BCOST method of constructing a confidence interval for the expected cost of a classifier.

BCOST(ρ,C,p,n)

Input: Confidence value ρ, cost matrix C, distribution p, number of test examples n,
number of bootstrap rounds N_b

for u **from** 1 **to** N_b **do**

Let $\tilde{M}_u = 0$, a simulated confusion matrix.

for v **from** 1 **to** n **do**

draw a pair (i, j) according to $p(i, j)$

increment $\tilde{M}_u(i, j)$.

end // for v

Let $\tilde{c}_u = \tilde{M}_u \cdot C$ be the cost of $\tilde{M}_u$

end // for u

Sort the $\tilde{c}_u$ values into ascending order.

Let $lb = \lfloor \frac{1-\rho}{2} \times N_b \rfloor + 1$; Let $ub = N_b + 1 - lb$

Output: The confidence interval $[\tilde{c}_{lb}, \tilde{c}_{ub}]$

end BCOST

5.2.2 *Comparing the Expected Cost of Two Classifiers:* BDELTACOST

The second statistical problem is to compare the costs of two different classifiers to decide which is better. Given two classifiers γ_1 and γ_2, we want to test the null hypothesis H_0 that the two classifiers have the same expected cost (on new test data) against the alternative hypothesis H_1 that the two classifiers have different costs. We want a test that will accept the null hypothesis with probability ρ if the null hypothesis is true (i.e., a test that has a Type I error of $1 - \rho$).

We follow essentially the same approach as for BCOST. The key is to define a new kind of three-dimensional confusion matrix M. The contents of cell $M(i_1, i_2, j)$ is the number of test set examples for which γ_1 predicted that they belong to class i_1, γ_2 predicted that they belong to class i_2, and their true class was j. Analogously, we can define a three-dimensional cost matrix Δ such that $\Delta(i_1, i_2, j) = C(i_1, j) - C(i_2, j)$. In other words, the value of $\Delta(i_1, i_2, j)$ is the amount by which the cost of classifier γ_1 is greater than the cost of classifier γ_2 when γ_1 predicts class i_1, γ_2 predicts class i_2, and the true class is j. Given a 3-D confusion matrix measured over a test set, we can compute the difference in the costs of γ_1 and γ_2 by taking the "dot product":

$$M \cdot \Delta = \sum_{i_1, i_2, j} M(i_1, i_2, j)\Delta(i_1, i_2, j).$$

We can obtain our statistical test by computing a confidence interval for $M \cdot \Delta$ and rejecting the null hypothesis H_0 if this confidence interval does not include zero. If the confidence interval does contain zero, we cannot reject the null hypothesis. The confidence interval is constructed in exactly the same way as for BCOST. We take the 3-D confusion matrix measured on the test set, normalize it, and apply a Laplace correction to get a multinomial probability distribution $p(i_1, i_2, j)$. We then simulate N_b confusion matrices $\tilde{M}_u$ by drawing (i_1, i_2, j) triples according to this distribution. We compute the difference in cost $\tilde{\delta}_u$ of each simulated confusion matrix

(by computing $\tilde{M}_u \cdot \Delta$), sort these differences, and choose the appropriate elements to construct the confidence interval. The pseudo-code for the BDELTACOST procedure is shown in Table 5.2.

TABLE 5.2: The BDELTACOST method for comparing two classifiers.

BDELTACOST(ρ,Δ,p,n)

Input: confidence value ρ, 3-D cost matrix Δ, distribution p, number of test examples n,
number of boostrap rounds N_b

for u **from** 1 **to** N_b **do**

Let $\tilde{M}_u = 0$, a simulated 3-D confusion matrix.

for v **from** 1 **to** n **do**

draw a pair (i_1, i_2, j) according to $p(i_1, i_2, j)$

increment $\tilde{M}_u(i_1, i_2, j)$.

end // for v

Let $\tilde{\delta}_u = \tilde{M}_u \cdot \Delta$ be the cost of $\tilde{M}_u$

end // for u

Sort the $\tilde{\delta}_u$ values into ascending order.

Let $lb = \lfloor \frac{1-\rho}{2} \times N_b \rfloor + 1$; Let $ub = N_b + 1 - lb$

Output: Reject H_0 with confidence ρ if $[\tilde{\delta}_{lb}, \tilde{\delta}_{ub}]$ does not contain 0;
Otherwise, do not reject H_0 with confidence ρ

end BDELTACOST

5.3 Summary

We have presented two new statistical tests for cost-sensitive evaluation of classifiers. The first procedure, BCOST, estimates the expected cost of a classification model. The second method, BDELTACOST, is a paired test for comparing two classifiers on a given domain. The two statistical tests described in this chapter are the first usable evaluation methods for cost-sensitive learning with a known cost matrix. These two tests were also reported in [121].

CHAPTER 6

EXPERIMENTAL RESULTS: COST-SENSITIVE LEARNING IN PRACTICE

6.1 Overview

This chapter presents the experiments that were performed to test the methods introduced thus far. We first assess the two evaluation tests introduced in Chapter 5. Next we test the cost sensitive learning methods from Chapters 3 and 4.

6.2 Evaluation of the Bootstrap Methods

This section presents experimental tests to verify that the two statistical evaluation methods described in chapter 5—BCOST and BDELTACOST—are working properly. We will also compare the confidence intervals these methods output to the confidence intervals based on the normal distribution.

6.2.1 Experimental Evaluation of BCOST

The purpose of this experiment is to construct a situation in which we have a classifier γ whose true expected average cost is known. We can then simulate a large number of different test sets, apply the BCOST procedure to each of them, and see whether the resulting confidence interval does indeed capture the true mean in fraction ρ of the trials. In our experiments, we set $\rho = 0.95$, so we are seeking a 95% confidence interval.

We performed our tests on two synthetic domains: *Expf5* and *Waveform*. Figure 6.1 shows the decision boundaries for the *Expf5* domain with two features (on the two axes) and five classes (denoted by the numbers in the figure). *Waveform* is a three-class problem that was invented with the purpose of testing CART by Breiman et. al [31]. Each of the classes represents a random convex combination of two waveform functions. The waveforms are triangular piecewise linear functions. Figure 6.2 shows h_1, one of the waveforms. The other two functions, h_2 and h_3, have similar shapes, but the waveforms are centered around 15 and 9, respectively. The data is sampled at the integers, and therefore has twenty-one attributes to which random noise was added. The attribute values of the instances from the three classes are generated using the following formulae. $y = 1$ (Class 1):

$$x_m = uh_1(m) + (1-u)h_2(m) + \epsilon_m, m = 1, \ldots m, \tag{6.1}$$

$y = 2$ (Class 2):

$$x_m = uh_1(m) + (1-u)h_3(m) + \epsilon_m, m = 1, \ldots m, \tag{6.2}$$

$y = 3$ (Class 3):

$$x_m = uh_2(m) + (1-u)h_3(m) + \epsilon_m, m = 1, \ldots m, \tag{6.3}$$

where u is a random number drawn from the uniform distribution, $0 \leq u \leq 1$, and the attribute noise ϵ_m is normally distributed with mean 0 and variance 1. The waveform data sets for our experiments are generated to be perfectly balanced, each class being equally represented.

In the case of *Expf5*, one million examples were drawn uniformly randomly from the space depicted in Figure 6.1 and labeled according to the decision boundaries in the figure. Similarly, for *Waveform*, one million test examples were generated using the mathematical model described above.

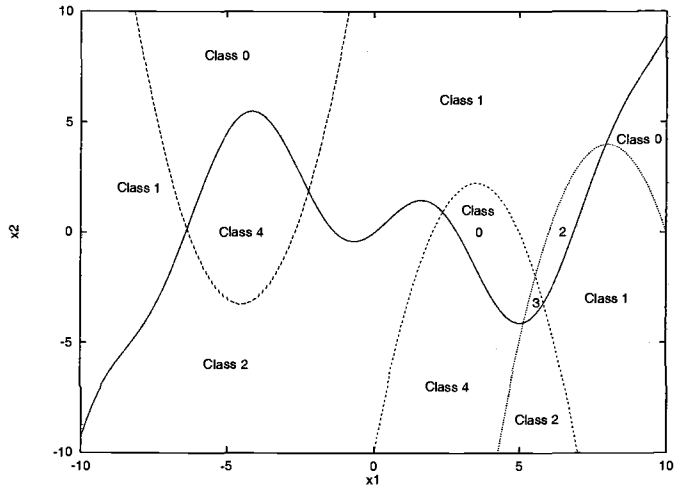


FIGURE 6.1: Decision boundaries for the *Expf5* data set.

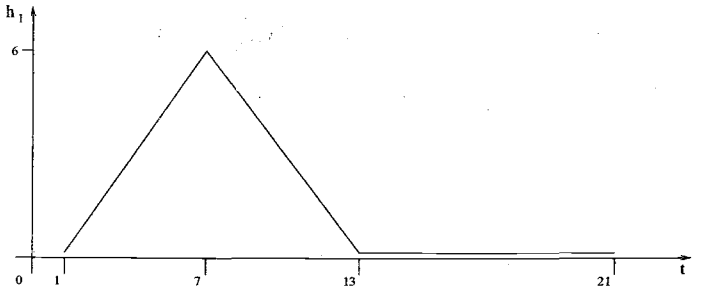


FIGURE 6.2: $h_1(t)$ - one of the three waveform functions employed to generate data for the waveform domain.

To build a classifier, we used the C4.5 algorithm [149] modified using the *AvgCost* procedure described in Chapter 3 (each example was weighted in proportion to the average value of the column of the cost matrix C corresponding to the label of the example). The classifier is trained on a separate set of one thousand examples drawn from the same distribution as the one million test examples. Table 6.1 shows the probabilities of each of the class labels computed from the test data.

TABLE 6.1: Class frequencies for the *Expf5* domain.

Class	Frequency (%)
0	21.02
1	44.73
2	25.68
3	0.16
4	8.41

Each experiment involves testing several different cost matrices, C. These were generated randomly based on eight different cost models. Table 6.2 shows the underlying distributions for each of the cost models. The diagonal elements of C are non-zero for cost models $M7$ and $M8$, and they are drawn from a uniform distribution over the range described in the third column. For all cost models, the off-diagonal elements are drawn from a uniform distribution over the range described in the second column of the table. In cost model $M5$, for example, the cost of mislabeling an example from class j as belonging to class i is determined by the ratio of the number of examples in class i to the number of examples in class j. In particular, if class i is very common, and class j is very rare, then this mistake will (on the average) be very expensive, because $P(i)/P(j)$ will be a large number. For cost model $M6$, we reversed this relationship, so that the least expensive errors are those that mislabel a rare class j as belonging to a common class i. Finally, model $M8$ is like model $M2$ except that there are non-zero costs on the diagonal of C. While in the case of *Expf5*, classes are imbalanced, and $P(i)/P(j)$ takes different values for each (i, j) pair.

TABLE 6.2: The cost models used for the experiments. Unif$[a, b]$ indicates a uniform distribution over the $[a, b]$ interval. $P(i)$ represents the prior probability of class i.

Cost Model	$C(i,j)$ $i \neq j$	$C(i,i)$
M1	Unif[0, 10]	0
M2	Unif[0, 100]	0
M3	Unif[0, 1000]	0
M4	Unif[0, 10000]	0
M5	Unif$[0, 1000 \times P(i)/P(j)]$	0
M6	Unif$[0, 1000 \times P(j)/P(i)]$	0
M7	Unif[0, 10000]	Unif[0, 1000]
M8	Unif[0, 100]	Unif[0, 10]

In *Waveform*, the three classes are equally represented, and therefore, cost models $M5$ and $M6$ are equivalent to $M3$ in this case.

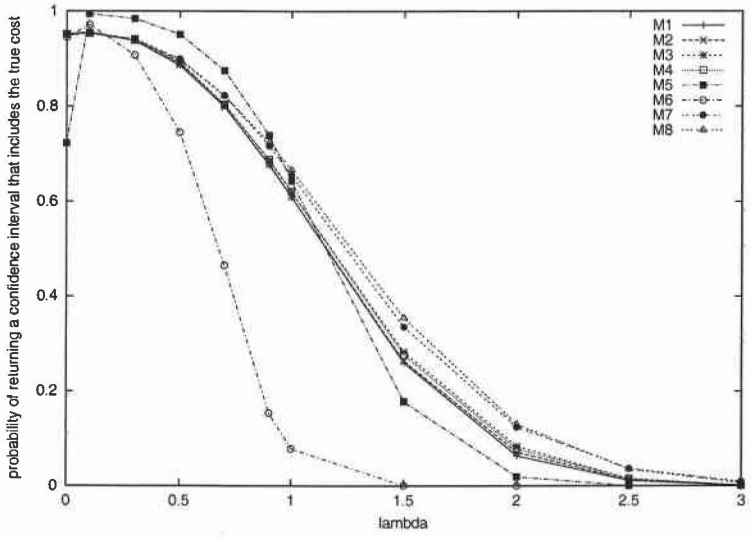
For each of the one thousand test sets, we ran BCOST and computed the 95% confidence interval for the expected cost. The true expected cost was considered to be the average cost for the entire one million example data set. We set the Laplace correction parameter $\lambda = 0.1$. Table 6.3 shows the number of test sets for which BCOST outputs a 95% confidence interval that includes the true expected cost. The values are averaged over ten cost matrices for each cost model. A perfect 95% confidence interval would include the true average cost exactly 0.95 N_b times out of N_b. In the case of *Waveform*, the confidence intervals are too narrow. For the *Expf5* domain, we see that the confidence intervals are all sufficiently wide, although the intervals for $M5$ and $M6$ are somewhat too wide.

We also computed a normal confidence interval for each test set both for *Expf5* and *Waveform*. For all five cost models, the true average cost was included in the normal 95% confidence interval 100% of the time (i.e., for all 1000 test sets). This means that the normal confidence intervals are definitely too wide.

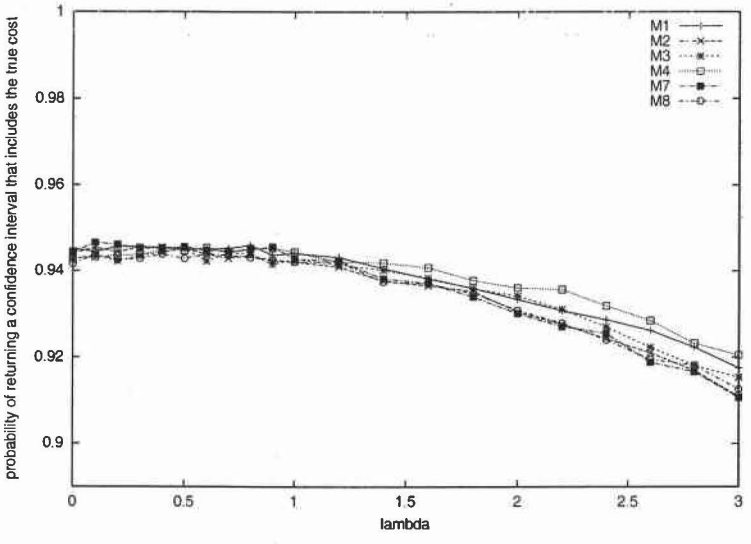
TABLE 6.3: Results of running BCOST on the 1000 1000-example test sets for nine different cost models. 10 cost matrices were used for each cost model. The second column shows the proportion of runs of BCOST for which the true cost was included in the confidence interval. 1000 bootstrap rounds were executed ($N_b = 1000$). Laplace correction $\lambda = 0.1$. True cost was computed as the average cost of the classifier for all 1,000,000 examples.

	Avg. # of cases for which the true cost was included in the c.i. (out of 1000)	
Cost Model	Expf5	Waveform
M1	0.9561	0.9444
M2	0.9555	0.9454
M3	0.9549	0.9436
M4	0.9539	0.9433
M5	0.9941	0.9436
M6	0.9717	0.9436
M7	0.9529	0.9465
M8	0.9531	0.9438

The sensitivity of BCOST to the choice of the Laplace correction parameter λ is shown in Figure 6.3. Each curve illustrates the probability that BCOST returns



(a) Expf5



(b) Waveform

FIGURE 6.3: Plots that describe the sensitivity of BCOST to the choice of λ for the nine cost models. For each cost model, 10 different matrices were tested, and each point plots the average result over the 10 matrices. In the case of *Expf5*, the test is significantly more sensitive to the choice of λ than in the case of *Waveform*.

a confidence interval that includes the true cost for one of the nine cost models, as λ varies from 0.0 to 3.0. In the case of *Expf5*, all curves reach a maximum within the interval (0.0,0.5]. No Laplace correction ($\lambda = 0$) will cause BCOST to generate confidence intervals that are too narrow, while values of λ larger than 0.5 will produce intervals that are too wide and biased too high. For *Waveform*, the test is significantly less sensitive to the choice of λ (note that all probability values from Figure 6.3(b) are between 0.92 and 0.95), but the curves never reach 0.95, which means that the test was more pessimistic. This proves that in this case the test gives confidence intervals that are marginally too narrow.

6.2.2 Experimental Evaluation of BDELTACOST

Given two arbitrary classifiers γ_1 and γ_2, the BDELTACOST procedure decides whether one classifier has a lower expected cost than another. There are only two possible decisions [50]: (1) the two algorithms have the same cost (the null hypothesis, H_0) and (2) the two algorithms have the different costs (the alternative hypothesis, H_1). Under these circumstances, any statistical test can commit two kinds of errors: Type I and Type II. A Type I error occurs when two classifiers have exactly the same cost, but the statistical test rejects the null hypothesis. If a statistical test is conducted with a confidence level of 0.95, then it should commit a Type I error with probability 0.05. The second kind of error, the Type II error, occurs when the statistical test fails to reject the null hypothesis even when the two classifiers have different expected costs.

For the assessment of BDELTACOST, we have run our experiments on the two domains described in the previous section: *Expf5* and *Waveform*. To evaluate the probability that BDELTACOST commits a Type I error, we face a difficult experimental problem. We need two classifiers, γ_1 and γ_2, that are different and yet that

have identical average costs. Furthermore, the worst case scenario for a statistical test usually occurs when the two classifiers are very different from each other [52], since this variability tends to "fool" the statistical test into thinking that the classifiers have different expected costs. So we want to design the two classifiers so that they misclassify different test examples and yet have the same average cost. To do this, we first ran C4.5 with *AvgCost* to produce γ_1. Then, we computed the confusion matrix M^* for γ_1 on the entire collection of one million test examples. To construct our second classifier, γ_2, we don't need to actually construct a decision tree (or any other real classifier)—all we need to do is assign a label to each test example, which we will declare to be the label assigned by γ_2. We want to assign these labels so that the cost of γ_2 (defined in this way) over the entire one million test examples is the same as the cost of γ_1. We will do this by ensuring that γ_2 has the same confusion matrix as γ_1. For each test example, $\langle \mathbf{x}_\ell, y_\ell \rangle$, we choose randomly a new label y_1 such that $M^*(y_1, y_\ell) > 0$, assign that label to example ℓ, and decrement $M^*(y_1, y_\ell)$. When we have labeled all of the test examples, every cell of M^* will be zero. This is equivalent to randomly permuting the γ_1 labels of all test examples belonging to the same true class. This makes the individual decisions of each classifier highly uncorrelated, but makes their confusion matrices identical, so their average cost on each of the 1000 test sets is the same. Table 6.4 shows the results of running this procedure on *Expf5* under each of our nine cost models with no Laplace correction. It plots the proportion of test sets for which the null hypothesis is not rejected. This number should be 0.950 if the test is working perfectly. The results show that the probability of Type I error is higher in the case of cost models that generate matrices with higher costs.

To test the sensitivity of BDELTACOST to the choice of the Laplace correction parameter, we ran the procedure described in the previous paragraph for different

TABLE 6.4: Results of running BDELTACOST on *Expf5* and *Waveform* on 1000-example test sets for the two classifiers that have the same expected average cost. The number of bootstrap rounds was 500 ($N_b = 500$). $\lambda = 0$.

	Proportion of cases for which H_0 was not rejected	
Cost Model	Expf5	Waveform
M1	0.94860	0.94780
M2	0.95016	0.94926
M3	0.94970	0.94710
M4	0.94853	0.94570
M5	0.93044	0.94710
M6	0.94620	0.94710
M7	0.94996	0.94410
M8	0.94592	0.94648

values of λ from the interval (0.0,0.5] on *Expf5*. $\lambda \geq 0.1$ caused BDELTACOST to generate confidence intervals that were too wide in the case of all cost models. In particular, for models $M5$ and $M6$ (in the case of *Expf5*), the probability of rejecting the null hypothesis for any $\lambda > 0$ was smaller than 0.005, which indicates that the confidence intervals were definitely too wide. This might happen because of the uniformity of the correction (λ is kept the same for all elements of the confusion matrix), and a non-uniform Laplace correction might be a better solution. One possibility is to set $\lambda(i_1, i_2, j)$ based on the distribution of elements in the confusion matrix or based on prior knowledge on the class distribution.

The standard way of measuring Type II errors is to plot a *power function* which graphs the probability that the null hypothesis will be rejected as a function of the amount by which the two classifiers differ in their expected costs [50].

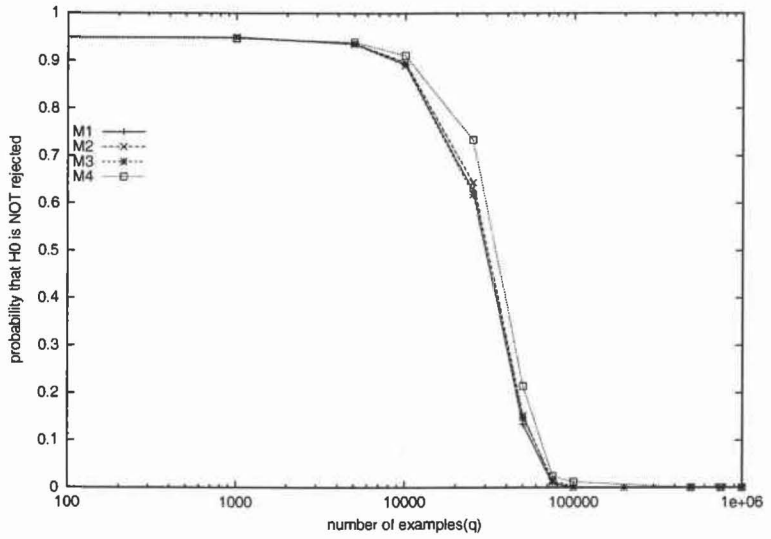
To generate the power curves, we must generate pairs of classifiers γ_1 and γ_2 that have a specified degree of difference in their expected costs. We started with the γ_1 and γ_2 classifiers constructed above. We then built a new classifier, γ_2' by randomly changing q of the labels that γ_2 had assigned to the test eamples. We varied q from 0 to 1,000,000, ran BDELTACOST 10 times for each value of q, and averaged the results for three matrices within each cost model.

Figures 6.4 and 6.5 plot these power functions for each of the nine cost models as a function of q (the nine curves have been split up for readability purposes) and Figures 6.9 and 6.10 plot the power functions as a function of the ratio in the cost of γ_2' to γ_1

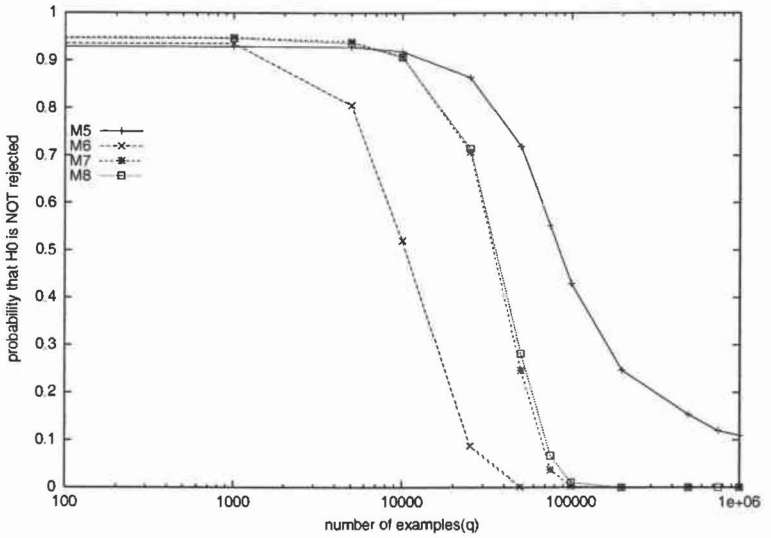
$$r_c = \frac{|cost(\gamma_2') - cost(\gamma_1)|}{cost(\gamma_1)}, \tag{6.4}$$

where $cost(\gamma)$ is the cost incurred by classifier γ when it classifies all the examples in a test set.

Figure 6.4 shows that in the case of cost model $M5$ and dataset *Expf5*, the test has a higher probability of Type II errors. That cost model assigns higher costs for misclassifying instances that belong to rare classes, and we can observe that in these two cases the probability that H_0 is not rejected is greater than 0 (0.092 for cost model $M5$) even when $q = 1,000,000$. Notice that while $M6$ looks best in Figure 6.4, it appears worst in Figure 6.9. This is because $M6$ assigns high costs to frequent errors, so even a small change in the number of such errors gives a high change in the cost ratio. For $M6$, BDELTACOST requires a large change in the cost ratio before it can discriminate between the two classifiers.

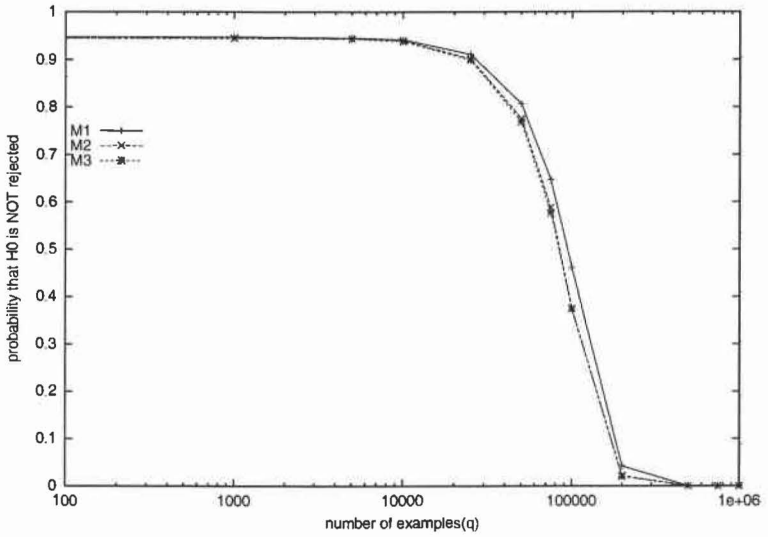


(a) Cost Models 1, 2, 3, and 4

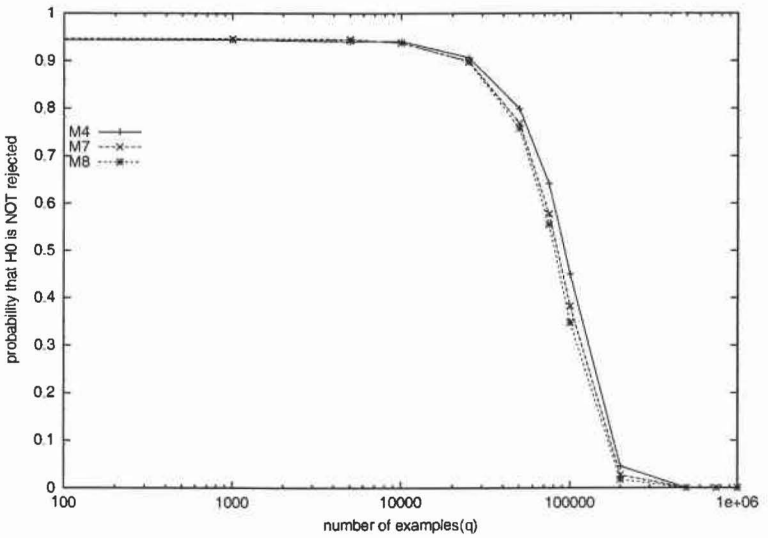


(b) Cost Models 5, 6, 7, and 8

FIGURE 6.4: Power of BDELTACOST ($\lambda = 0$) for *Expf5*. Each curve plots the probability of rejecting H_0 as a function of q.



(a) Cost Models 1, 2, and 3,



(b) Cost Models 4, 7, and 8

FIGURE 6.5: Power of BDELTACOST ($\lambda = 0$) for *Waveform*. Each curve plots the probability of rejecting H_0 as a function of q.

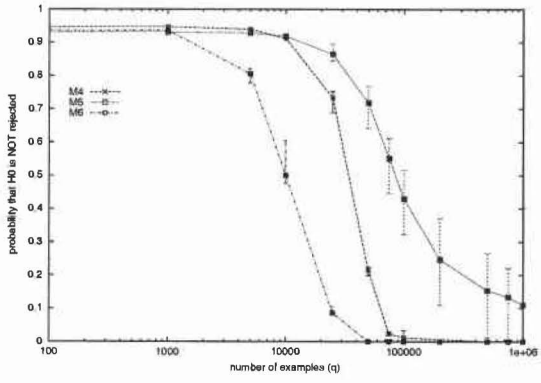
We can also look at the variability of the probability of rejecting the null hypothesis, within each cost model. For readability purposes this information is plotted in separate graphs shown in Figures 6.6, 6.7, and 6.8. Each graph uses error bars to show the range of measured values. The variability is minor except for cost model $M5$ when q is large.

In addition to running BDELTACOST on each pair of classifiers, we also ran the paired-differences z test as follows. Let c_ℓ^1 be the cost incurred by classifier γ_1 when it classifies example ℓ, and c_ℓ^2 be the cost incurred by classifier γ_2', when it classifies example ℓ. The paired-differences test constructs a normal 95% confidence interval for the differences $c_\ell^1 - c_\ell^2$ and rejects the null hypothesis if this confidence interval does not contain zero.

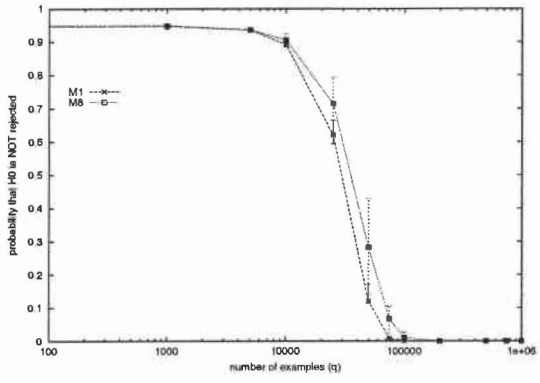
When we ran this test for the various cost models and the various values of q for both *Expf5* and *Waveform*, we found that in all cases, the null hypothesis was never rejected. In other words, this test has no power to detect when one classifier has different expected cost than another.

Next, we have tested the sensitivity of BDELTACOST to the number of bootstrap rounds, N_b. We have repeated all experimental tests described above for $N_b = 300$ and $N_b = 100$ both on *Expf5* and on *Waveform*. Figures 6.11, 6.12, 6.13, and 6.13 show the first portion ($q \leq 10000$) of the power curves for each cost model for the three different values of N_b: 100, 300, and 500. We have chosen to show only the first portions of the curves, because that is the region where, in some cases, the setting of N_b has influenced the tests. For larger values of q, there were no significant differences between the curves for most of the cost models.

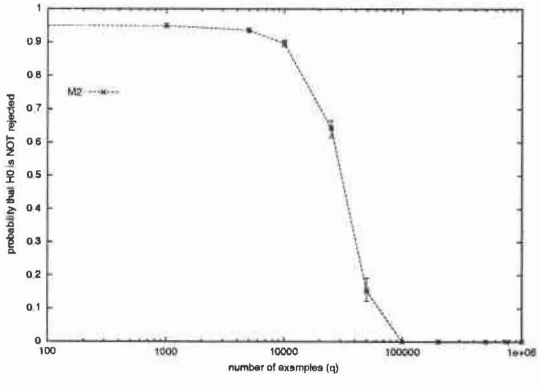
For both *Expf5* and *Waveform*, setting the number of bootstrap rounds, N_b, to 100 has caused BDELTACOST to output intervals that are significantly more narrow than for $N_b = 300$ or $N_b = 500$. In the meantime, setting N_b to 500 has produced



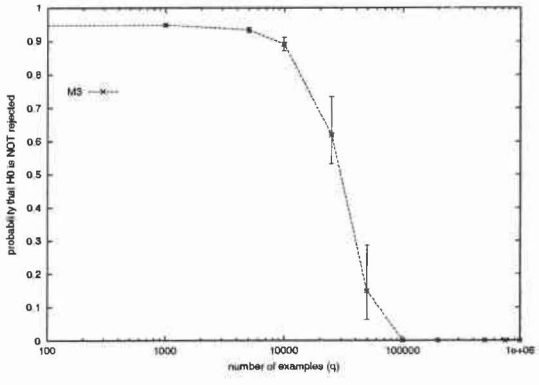
(a) Cost Models 4, 5, and 6



(b) Cost Models 1 and 8

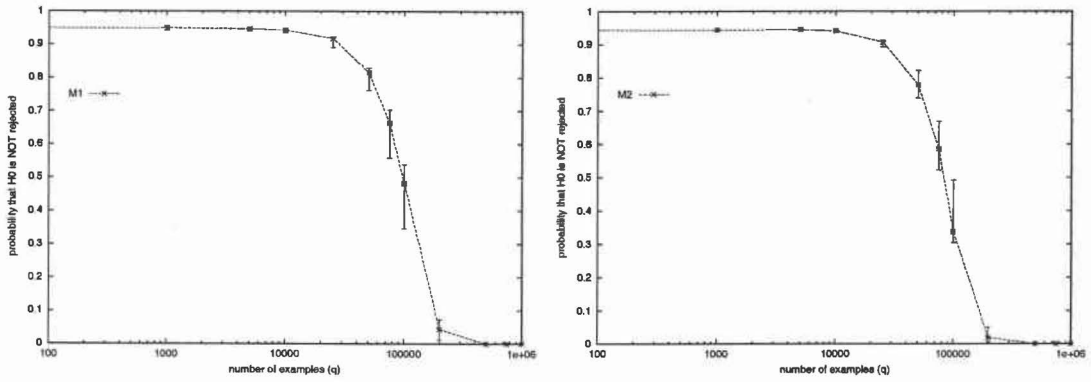


(c) Cost Model 2



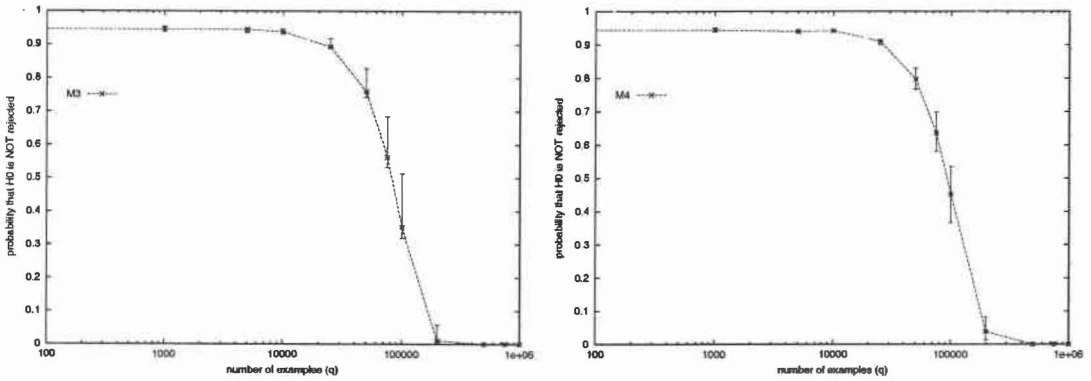
(d) Cost Model 3

FIGURE 6.6: Enlarged plots from Figure 6.4. Error bars show range of observed probability of rejecting H_0. $N_b = 500$. $\lambda = 0$.



(a) Cost Model 1

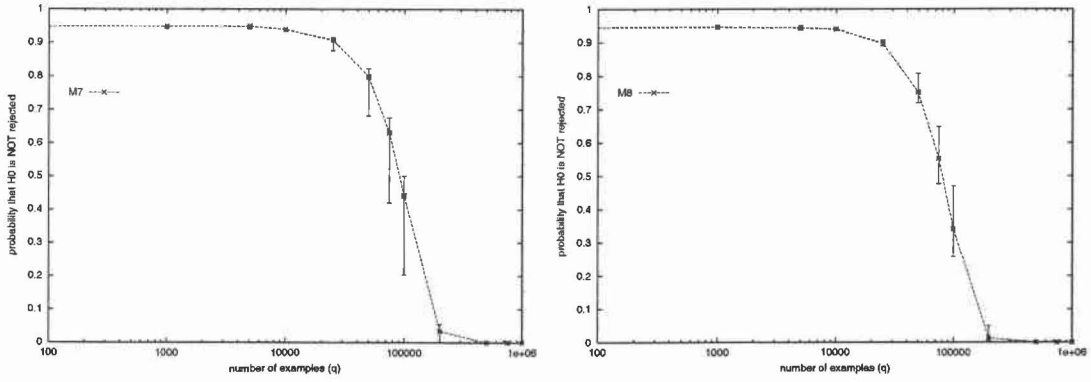
(b) Cost Model 2



(c) Cost Model 3

(d) Cost Model 4

FIGURE 6.7: Enlarged plots from Figure 6.5 for Cost Models 1, 2, 3, and 4. Error bars show range of observed probability of rejecting H_0. $N_b = 500$. $\lambda = 0$.



(a) Cost Model 7

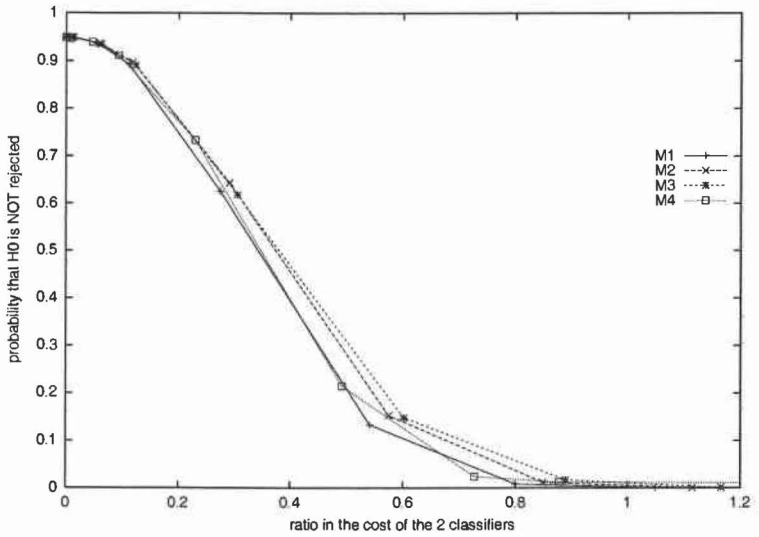
(b) Cost Model 8

FIGURE 6.8: Enlarged plots from Figure 6.5 for Cost Models 7 and 8. Error bars show range of observed probability of rejecting H_0. $N_b = 500$. $\lambda = 0$.

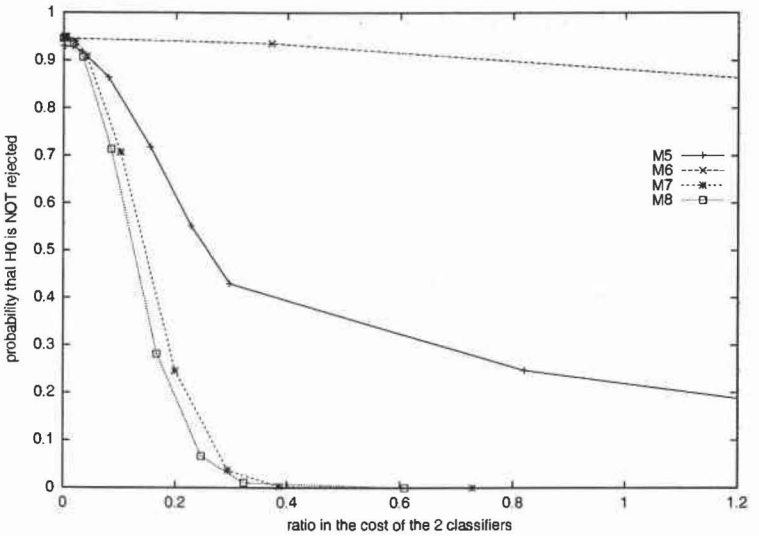
slightly (but only slightly) better confidence intervals in comparison to $N_b = 300$ for all cost models except $M8$ on *Expf5*. These results suggest that if classifier testing time is a major concern for an application, BDELTACOST with 300 bootstrap rounds is a reasonable choice for a test procedure.

Finally, we wanted to analyze how sensitive the BDELTACOST test is to the distribution represented by M^* (the confusion matrix), or equivalently, to the accuracy of the classifier. C4.5 with *AvgCost* builds fully grown, cost-sensitive decision trees, and, as a result, M^* and p (the probability matrix computed using the counts in M^*) have larger values on the diagonal.

We reran all experiments on a classifier that is just slightly better than random guessing — the cost-insensitive, decision tree algorithm with a maximum depth of the model set to two. Figures 6.15 and 6.16 show the power curves of BDELTACOST for each cost model, for depth-two decision trees. It is important to observe that in

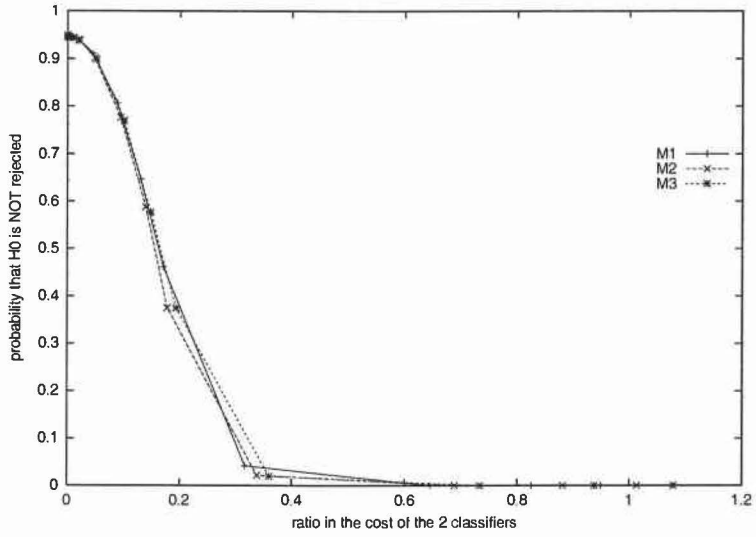


(a) Cost Models 1, 2, 3, and 4

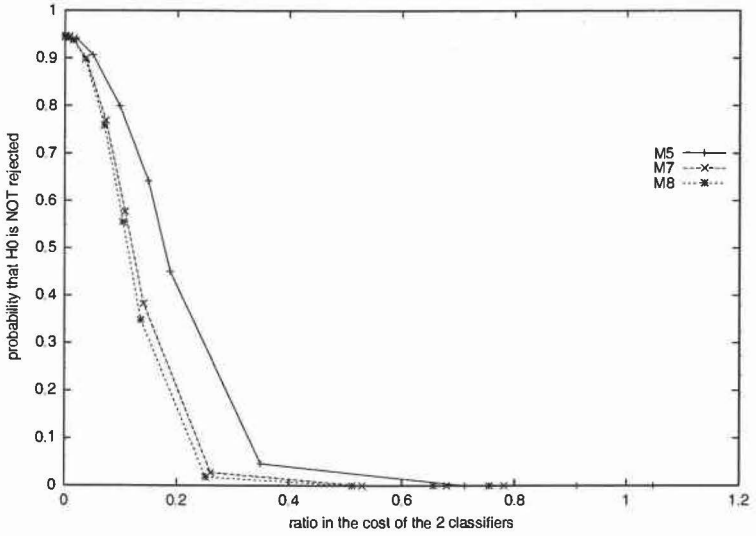


(b) Cost Models 5, 6, 7, and 8

FIGURE 6.9: Power of BDELTACOST ($\lambda = 0$) for *Expf5*. Each curve plots the probability of rejecting H_0 as a function of the cost ratio.

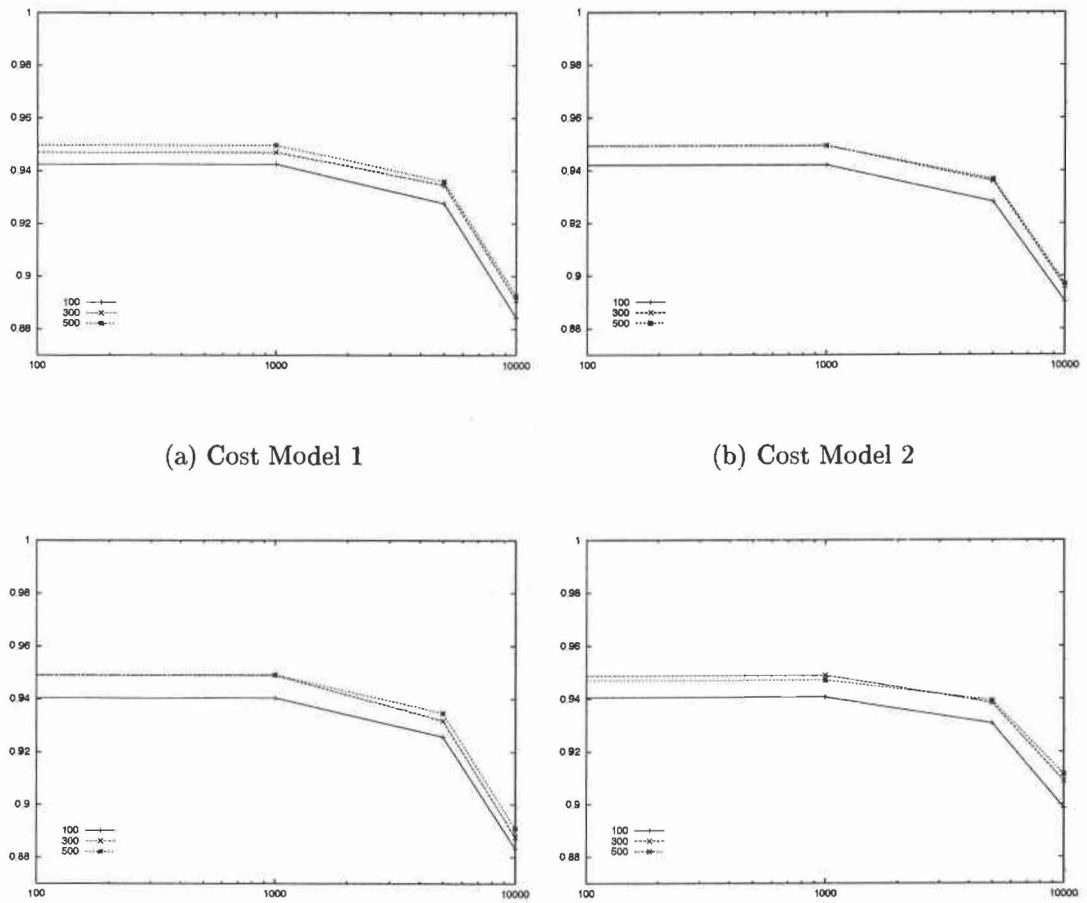


(a) Cost Models 1, 2, and 3



(b) Cost Models 5, 7, and 8

FIGURE 6.10: Power of BDELTACOST ($\lambda = 0$) for *Waveform*. Each curve plots the probability of rejecting H_0 as a function of the cost ratio.



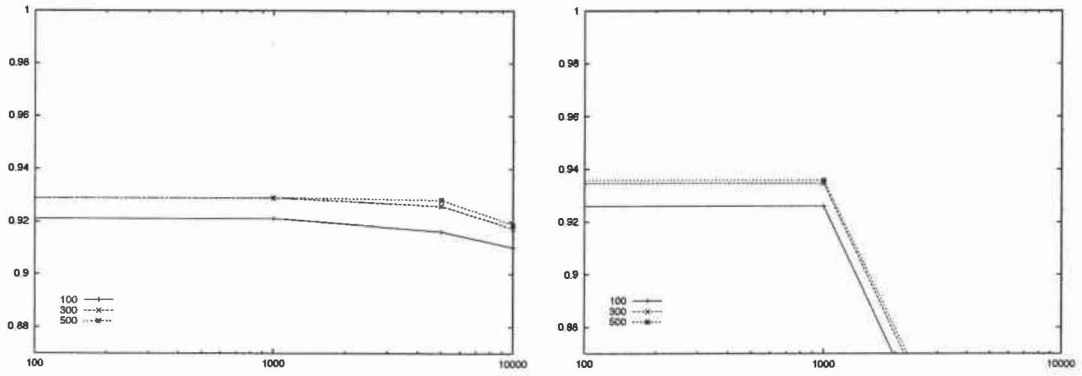
(a) Cost Model 1

(b) Cost Model 2

(c) Cost Model 3

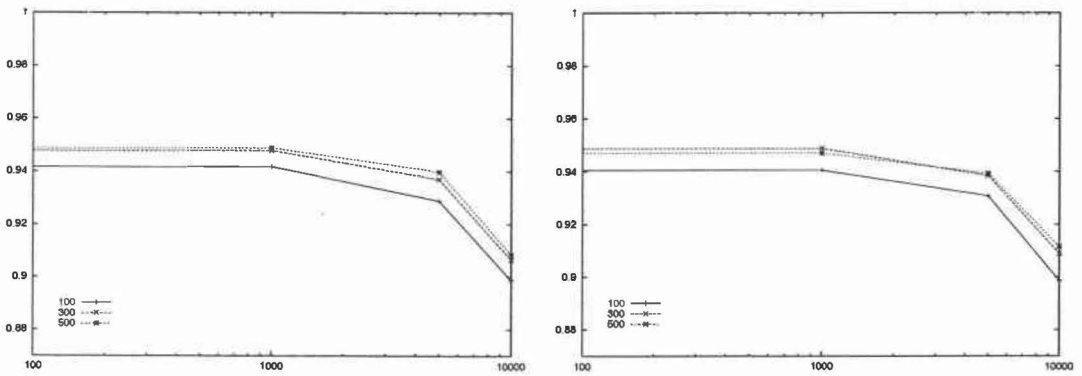
(d) Cost Model 4

FIGURE 6.11: Power of BDELTACOST ($\lambda = 0$) for *Expf5*. Each plot depicts the power curves for a cost model, for $N_b = 100$, $N_b = 300$ and $N_b = 500$. Cost models 1, 2, 3, and 4.



(a) Cost Model 5

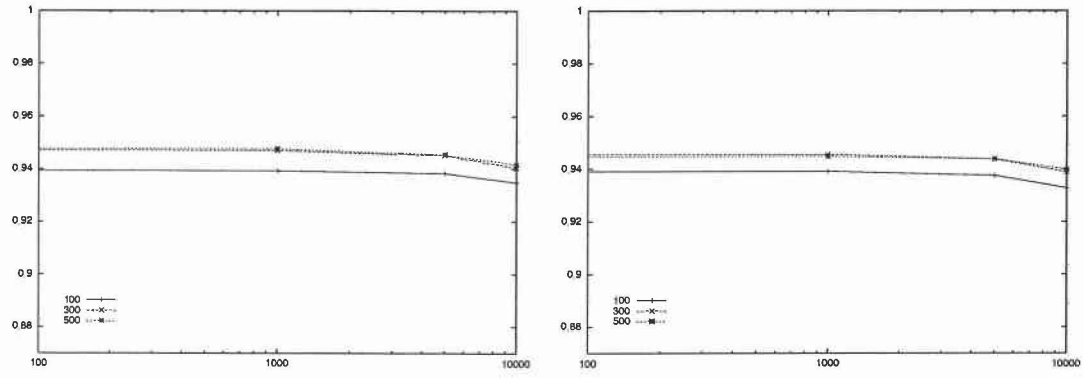
(b) Cost Model 6



(c) Cost Model 7

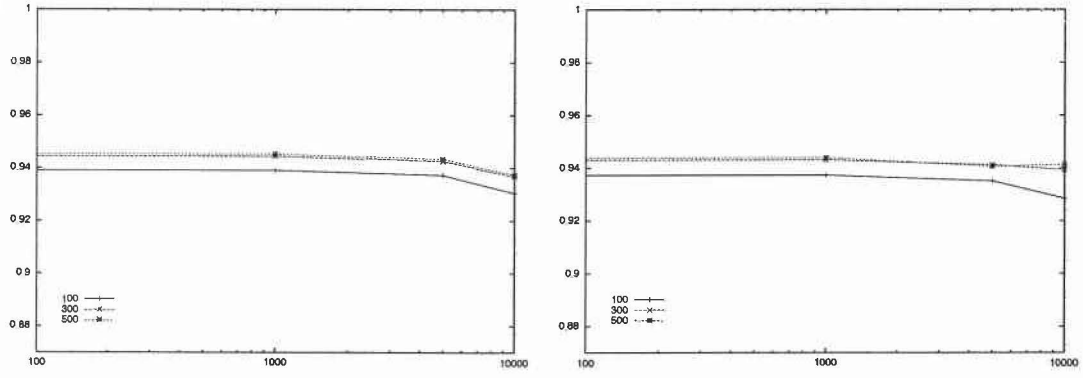
(d) Cost Model 8

FIGURE 6.12: Power of BDELTACOST ($\lambda = 0$) for *Expf5* for different values of N_b. Cost models 5, 6, 7, and 8.



(a) Cost Model 1

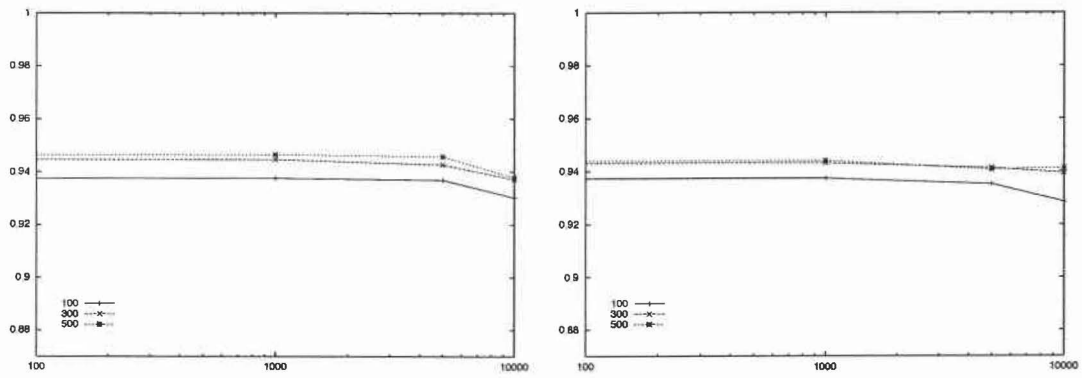
(b) Cost Model 2



(c) Cost Model 3

(d) Cost Model 4

FIGURE 6.13: Power of BDELTACOST ($\lambda = 0$) for *Waveform*. Each plot depicts the power curves for a cost model, for $N_b = 100$, $N_b = 300$ and $N_b = 500$.



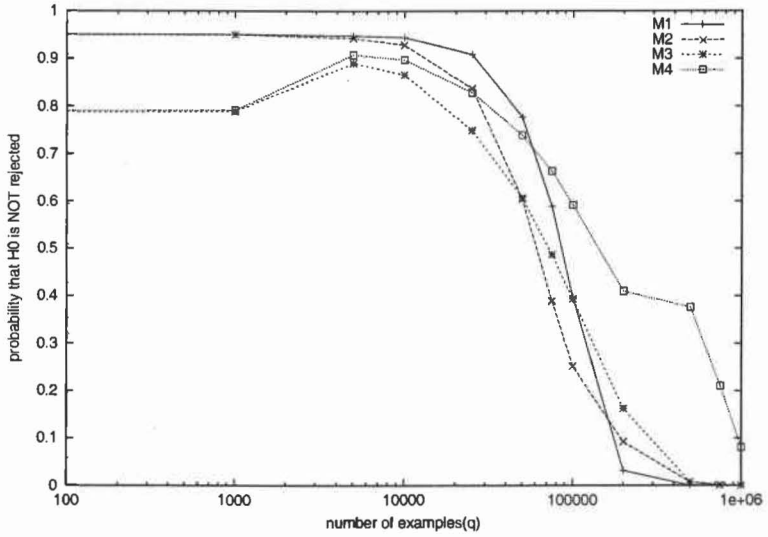
(a) Cost Model 7

(b) Cost Model 8

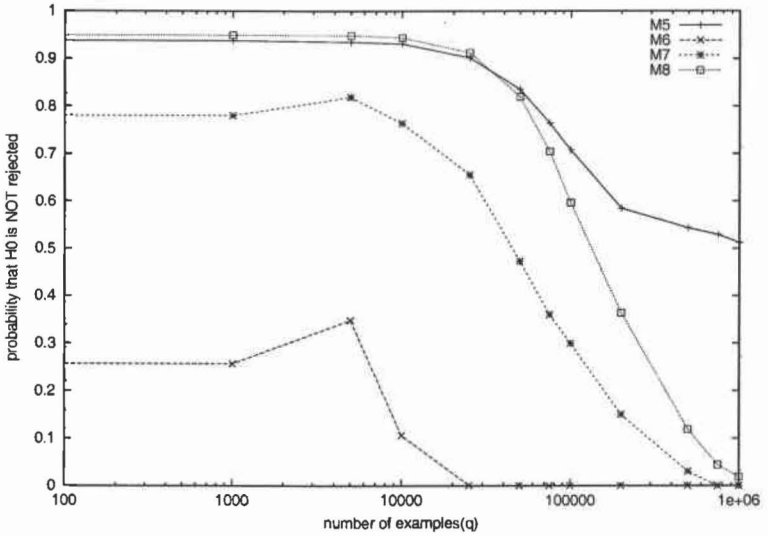
FIGURE 6.14: Power of BDELTACOST ($\lambda = 0$) for *Waveform*. Each plot depicts the power curves for a cost model, for $N_b = 100$, $N_b = 300$ and $N_b = 500$.

the case of *Expf5*, BDELTACOST falsely rejects the null hypothesis for cost models with high off-diagonal costs. Especially in the case of $M6$, the poor predictions of the classifiers combined with the high penalties for misclassifying the most common class labels have caused the algorithm to exhibit a high Type I Error. The test was less sensitive to the classification algorithm in the case of *Waveform* because of the uniform class distribution. For both domains, BDELTACOST with depth-two trees had a slightly higher Type II Error compared to the cost-sensitive, fully grown decision trees.

Overall, the tests show that the very high class imbalance of *Expf5* combined with cost functions that penalize heavily misclassifying either rare or more common instances, can cause the bootstrap methods to have higher variance and instability. The behavior of the tests on *Waveform* is significantly more stable and less sensitive to the input parameters.

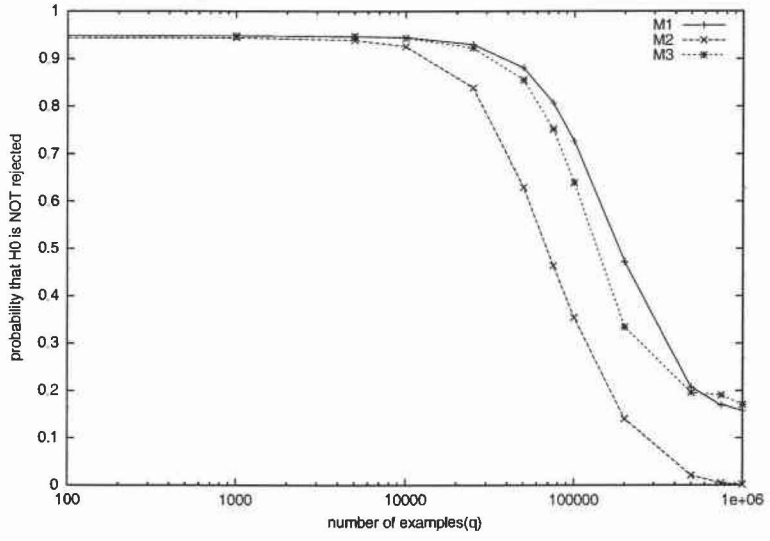


(a) Cost Models 1, 2, 3, and 4

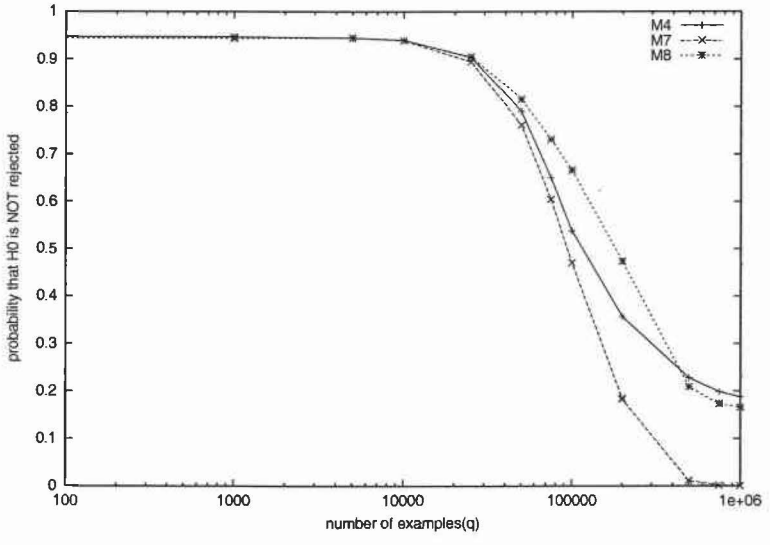


(b) Cost Models 5, 6, 7, and 8

FIGURE 6.15: Power of BDELTACOST ($\lambda = 0$) for *Expf5* for depth-two decison tree classifiers. Each curve plots the probability of rejecting H_0 as a function of q.



(a) Cost Models 1, 2, and 3,



(b) Cost Models 4, 7, and 8

FIGURE 6.16: Power of BDELTACOST ($\lambda = 0$) for *Waveform* for depth-two decision tree classifiers. Each curve plots the probability of rejecting H_0 as a function of q.

TABLE 6.5: Data sets studied in our experiments.

Name	# of Classes	# of Attributes	Discrete	Continuous
Abalone	21	8	1	7
Audiology (audio)	8	69	69	0
Breast Cancer Yugoslavia (bcy)	2	9	5	4
Glass	6	9	0	9
Hepatitis	2	19	13	6
Iris	3	4	0	4
Liver Disorders (liver)	2	6	0	6
Lung Cancer (lung)	3	56	56	0
Lymphography (lympho)	4	19	16	3
Segmentation (segment)	7	19	0	19
Sonar	2	60	0	60
Soybean large (soybean)	15	35	35	0
Vehicle	4	18	0	18
Voting records (voting)	2	16	16	0
Wine	3	13	0	13
Zoology (zoo)	7	16	15	1

6.3 Evaluation of Algorithms for Data Manipulation

Now that we have validated the statistical tests, we apply them to evaluate the cost-sensitive learning methods introduced in Chapters 3 and 4.

We have first tested the methods for cost-sensitive learning by training data manipulation (described in Chapter 3). To do this, we have employed data from a set of sixteen domains of the the UC Irvine machine learning repository [18].

TABLE 6.6: Approximate class distributions of the studied datasets.

Name	# of Classes	Class Distribution
abalone	21	[.003,.013,.03,.06,.09,.14,.17,.15,.12,.063,.047,.03,.024,.016, .014,.01,.008,.006,.003,.001,.002]
audio	8	[.27,.11,.1,.28,.04,.03,.12,.05]
bcy	2	[.7,.3]
glass	6	[.33,.36,.08,.06,.04,.13]
hepatitis	2	[.21,.79]
iris	3	[.33,.33,.33]
liver	2	[.42,.58]
lung	3	[.28,.4,.32]
lympho	4	[.01,.55,.42,.02]
segment	7	[.14,.14,.14,.14,.14,.14,.14]
sonar	2	[.53,.47]
soybean	15	[.031,.031,.031,.14,.07,.031,.031,.15,.031,.031,.031,.07,.031,.14,.14]
vehicle	4	[.254,.254,.238,.254]
voting	2	[.613,.387]
wine	3	[.33,.40,.27]
zoo	7	[.405,.197,.05,.129,.04,.08,.099]

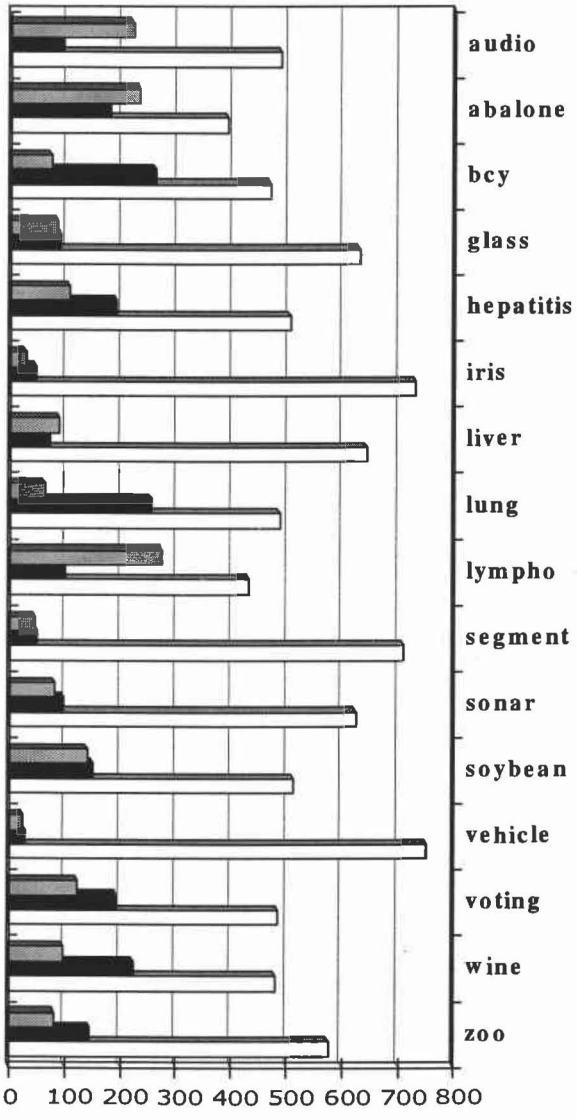


FIGURE 6.17: Barplot showing the results of running BDELTACOST for comparing the *Powell10* wrapper method against *AvgCost* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *AvgCost*, and the grey bars represent the number of wins for *Powell10*.

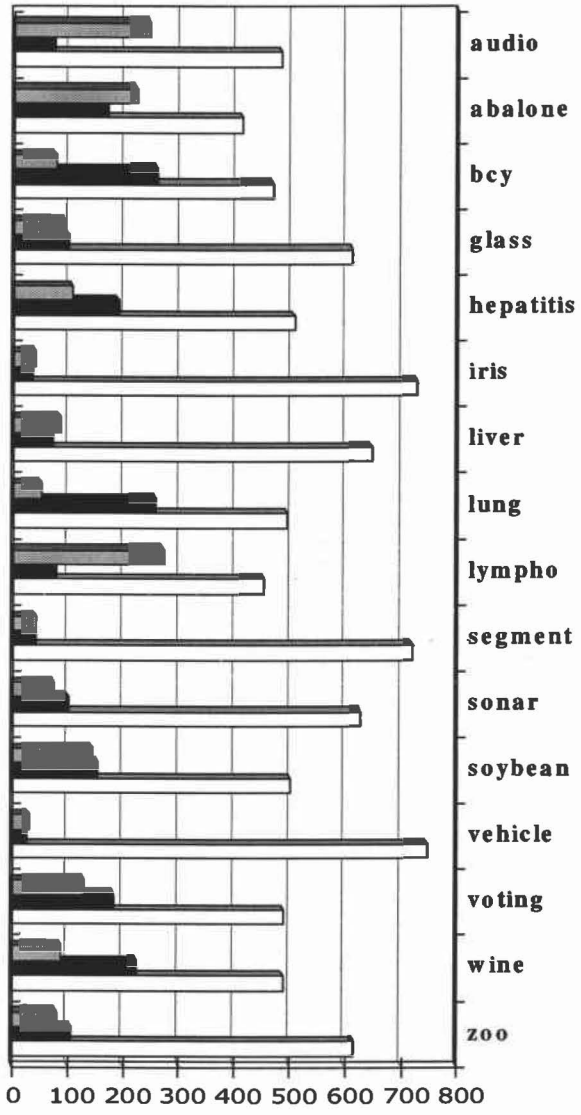


FIGURE 6.18: Barplot showing the results of running BDELTACOST for comparing the *Powell10* wrapper method against *MaxCost* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *MaxCost*, and the grey bars represent the number of wins for *Powell10*.

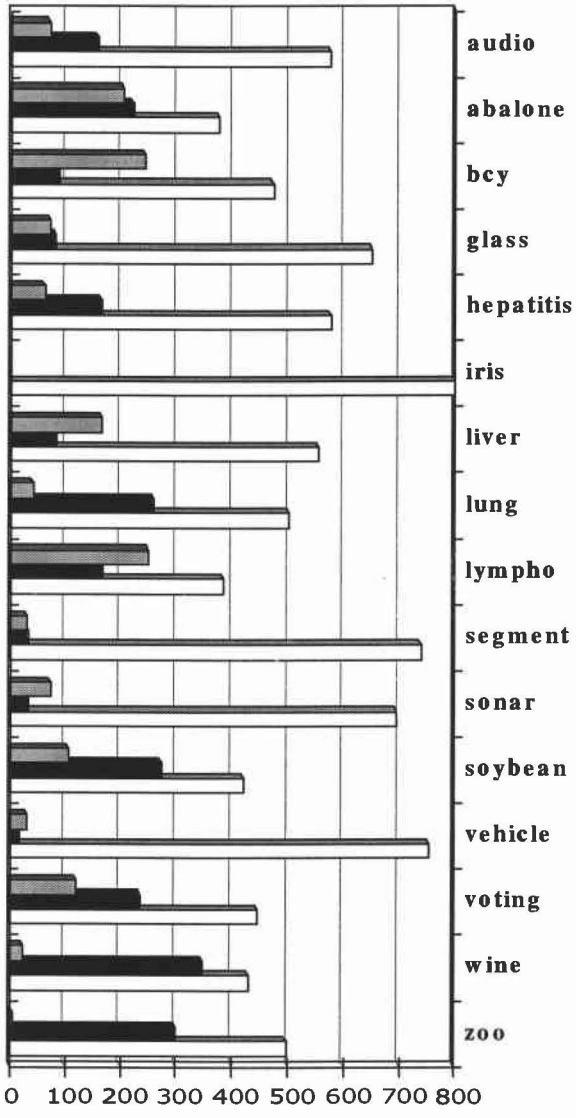


FIGURE 6.19: Barplot showing the results of running BDeltaCost for comparing the *Powell10* wrapper method against *ClassFreq* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *ClassFreq*, and the grey bars represent the number of wins for *Powell10*.

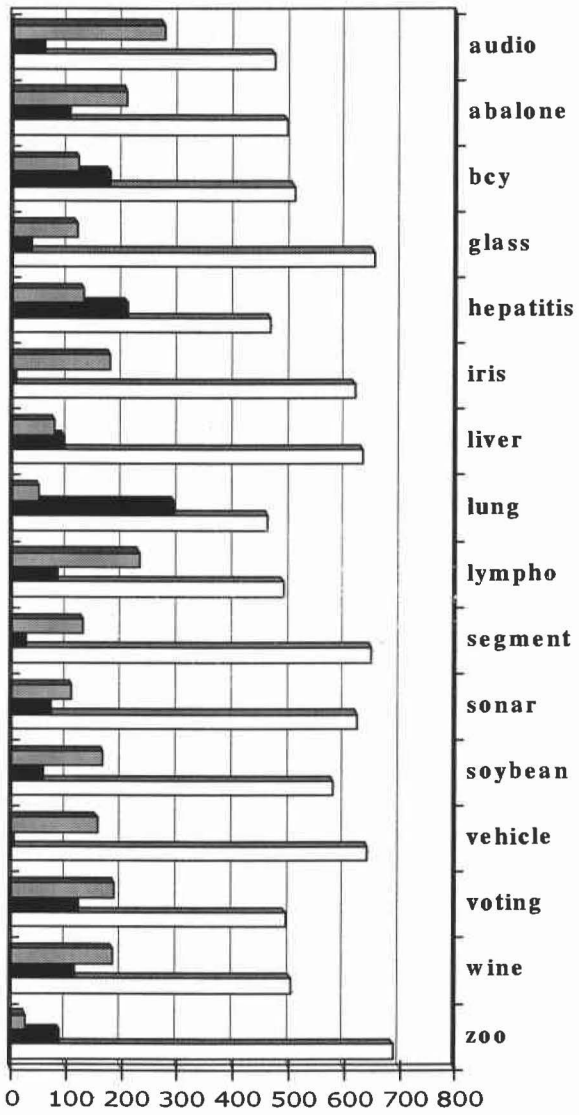


FIGURE 6.20: Barplot showing the results of running BDELTACOST for comparing the *Powell10* wrapper method against *EvalCount10* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *EvalCount10*, and the grey bars represent the number of wins for *Powell10*.

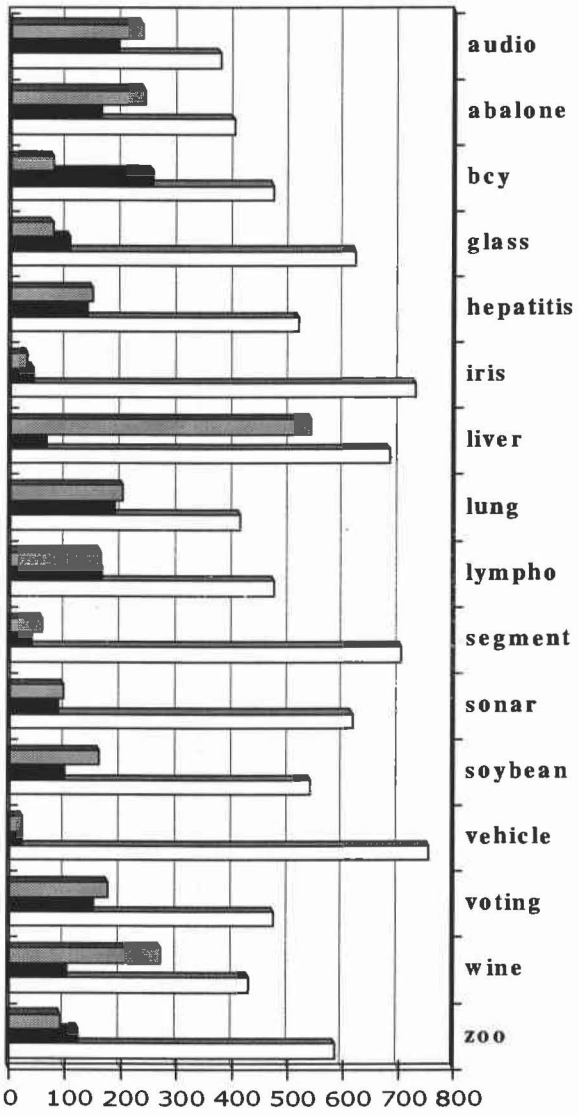


FIGURE 6.21: Barplot showing the results of running BDELTACOST for comparing the *Powell20* wrapper method against *AvgCost* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *AvgCost*, and the grey bars represent the number of wins for *Powell20*.

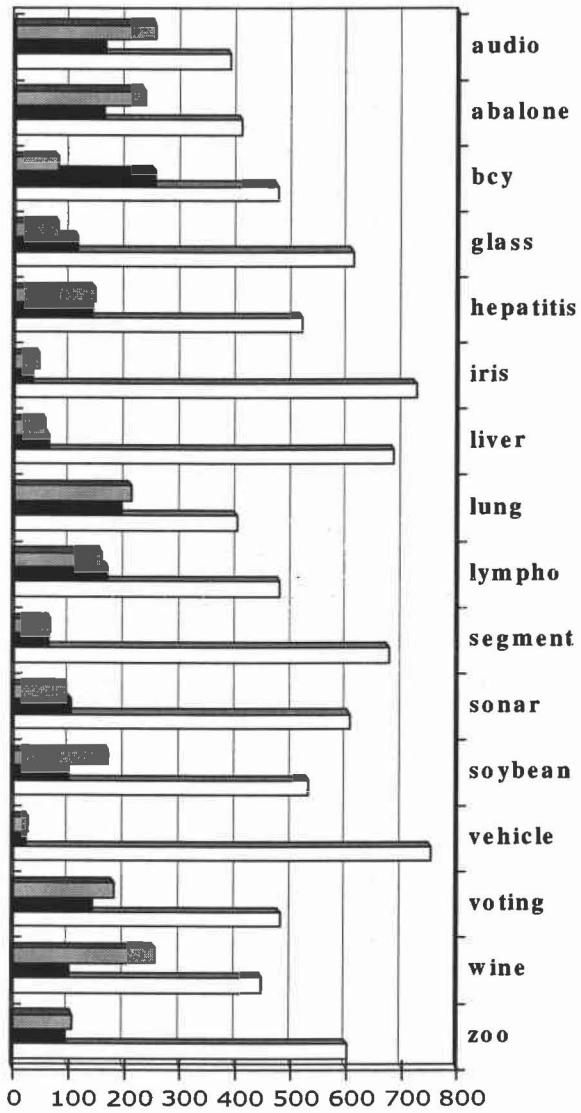


FIGURE 6.22: Barplot showing the results of running BDELTACOST for comparing the *Powell20* wrapper method against *MaxCost* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *MaxCost*, and the grey bars represent the number of wins for *Powell20*.

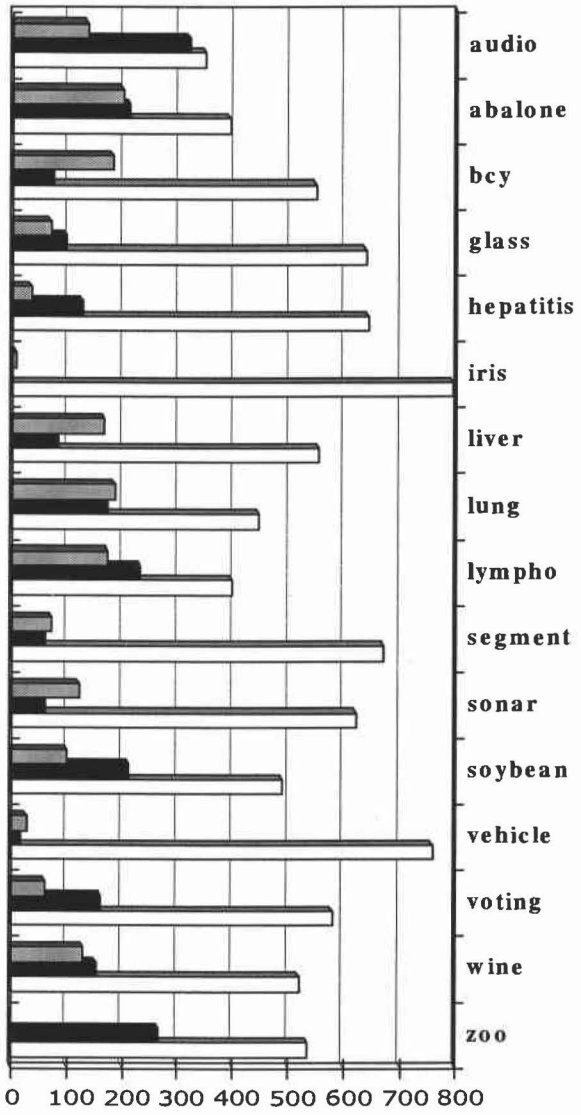


FIGURE 6.23: Barplot showing the results of running BDELTACOST for comparing the *Powell20* wrapper method against *ClassFreq* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of *ClassFreq*, and the grey bars represent the number of wins for *Powell20*.

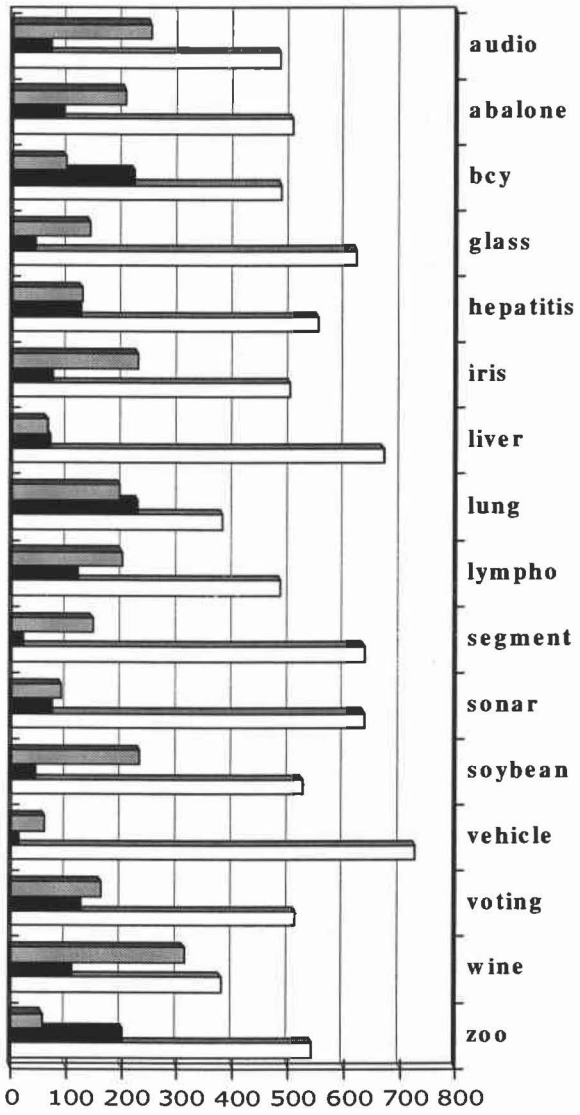


FIGURE 6.24: Barplot showing the results of running BDELTACOST for comparing the *Powell20* wrapper method against *EvalCount20* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of it EvalCount20, and the grey bars represent the number of wins for *Powell20.*

These datasets are presented in Table 6.5. The tasks associated with them are described in Appendix A. The class distributions for each of the domains are given in Table 6.6. We have run ten-fold cross validation for all eight cost models described by the distributions in Table 6.2. For each of the models we have generated ten cost matrices. To do the testing we have used BDELTACOST. For each of the domains, 800 tests were performed (8 cost models, 10 matrices per model, ten-fold cross-validation).

Figures 6.17, 6.18, and 6.19 compare the *Powell10* wrapper method against *AvgCost*, *MaxCost*, and *ClassFreq*. Figures 6.21, 6.22, and 6.23 compare the *Powell20* wrapper method against *AvgCost*, *MaxCost*, and *ClassFreq*. Figures 6.20 and 6.24 compare the two methods that use hold-out data for validation, *Powell* and *EvalCount*, for 10% and 20% (denoted as *Powell10*, *Powell20*, *EvalCount10*, and *EvalCount20*, respectively) of the available data used for validation.

The charts show that for all tests, and for all domains, the number of ties between the tested methods in the pairwise comparisons is always larger than the number of wins of either of them. *Powell20* performs slightly better than *Powell10*. *Powell20* has a significantly better performance than *AvgCost* only in the case of liver and wine domains, and a better performance than *EvalCount20* for audio, soybean, and wine. In the meantime it is interesting to observe that *MaxCost* is worse than *AvgCost*, although their nature is very similar. The methods that give the best results are *ClassFreq* and *Powell20*. When these two methods are compared (see Figure 6.23), *ClassFreq* outperforms *Powell20* on seven data sets (audio, hepatitis, lymphography, soybean, voting records, wine and zoo), *Powell20* wins on three domains (breast cancer Yugoslavia, liver disorders, and sonar), while on the rest of the domains the two methods are tied (based on the 95% confidence interval).

Given that the wrapper methods are expensive in terms of processor time, these results suggest that the time required by them to compute the weight parameters, might not always be worthwhile. In many cases, a simpler and faster heuristic method like *ClassFreq* can give even better results.

An important thing to observe is that among the heuristic methods, the best results are obtained by *ClassFreq*, the algorithm that in most of the cases sets the smallest values on the weights, while *EvalCount20* and *MaxCost*, the algorithms that set the highest weights, give the worst results. Even in the case of two-class problems (for which *MaxCost* is equivalent to the traditional weighting according to the cost of misclassification for a class), *MaxCost* is outperformed by *ClassFreq*.

6.4 Evaluation of the Probability Estimation Algorithms

To test the accuracy of the class probability estimates computed by the bagged LOTs (B-LOTs) and by the bagged PETs (B-PETs), we first need a synthetic domain for which the true probabilities are known. As pointed out by Provost and Domingos [144], one of the hardest tasks for class probability learning are the ones in which the data was generated by overlapping distributions. Therefore, for our test, we have employed a two-class domain in which the examples were generated by the two overlapping Gaussians (each Gaussian corresponds to one class) shown in Figure 6.25.

We have run the B-LOTs and the B-PETs on data from the overlapping Gaussians domain, and the estimated values of the probabilities are plotted against the actual values in the scatter plots from Figures 6.26, 6.27, 6.28, 6.29, 6.30, 6.31, 6.32 and 6.33. Perfect probability estimates correspond to the points on the diagonal (which is plotted in all graphs, for reference). To analyze the sensitivity of the algorithms to the different parameters, we varied the size of the training data ($TDsize$) between 100 and 1000, and the minimum number of instances ($MINinst$) of a tree leaf from

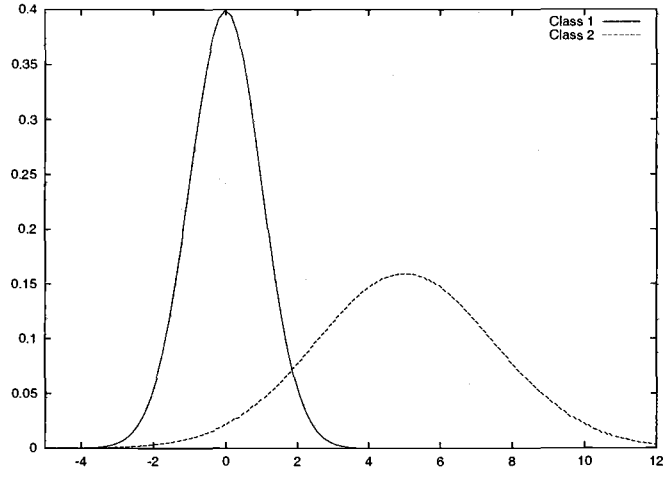


FIGURE 6.25: The overlapping Gaussians domain.

2 to 12. For all tests we plot the probabilities for 1000 test points (the same test data was used throughout these experiments).

From these plots we can observe that the B-PETs have computed very good rankings when the minimum number of instances in a leaf was large (especially for $TD = 300$). However, we can observe the flat regions corresponding to large number of instances being assigned the same probability. For small values of $MINinst$ (e.g., $MINinst = 2$) or large values of TD (e.g.,$TD = 600$) we can observe that there are some oscillations that disrupt the flat regions in a wrong direction. In the case of B-LOTs, small values of TD and $MINinst$ have produced some extreme estimates at both ends of the $[0, 1]$ interval. When TD is larger, the lazy learning mechanism causes some of these extreme values to be "dragged" towards the diagonal. In our view the best estimates are computed for $TD = 300$ and $MINinst = 8$, and for $TD = 1000$ and $MINinst = 12$. Although the flat regions tend to disappear as TD is assigned larger values, it is clear that the estimates are far from perfect.

Our intuition is that the purity-based criterion which is used for the LOTs and B-LOTs is one of the causes for the bad estimates.

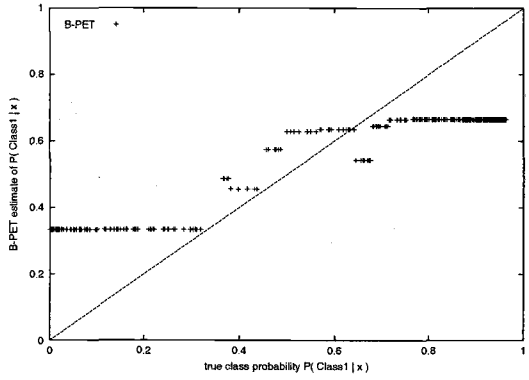
The good rankings of the B-PETs seem to be affected by the Laplace correction especially in regions that are close to 0 and 1. To test the influence of the Laplace correction parameter, we set it to 0 and reran the B-PET algorithm. Figures 6.34 and 6.35 show the scatter plots for $MINinst = 2$ and $MINinst = 8$. We can observe that, although the rankings are good (especially for large values of TD and $MINinst$), the probability estimates are very inaccurate.

Next, we have tested the lazy class probability estimators and the B-PETs on the UC Irvine datasets presented in Table 6.5.

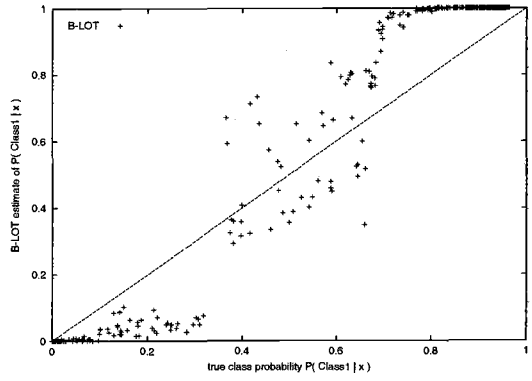
Again, we have run ten-fold cross validation for all eight cost models described by the distributions in Table 6.2. This time, for each of the models we have generated twenty-five cost matrices.

Figure 6.36 compares the B-LOTs and the B-PETs and shows the number of times one procedure outperformed the other and the number of ties (computed by BDELTACOST based on the 95% confidence interval). A total number of two thousand bootstrap tests were performed for each of the domains — two hundred (eight cost models, twenty-five matrices per model) times ten (the number of cross validation folds).

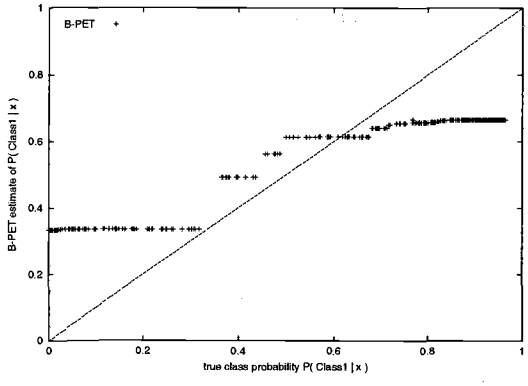
For all tasks except the lung-cancer domain, the number of wins for the B-LOTs is significantly larger than the number of B-PETs wins. In the case of nine domains (audiology, abalone, iris, lymphography, segmentation, soybean, voting-records, wine and zoo), the B-LOTs were the clear winner. These results suggest that the B-LOTs work very well in these application domains. The question is, "*What* makes them to outperform the B-PETs?".



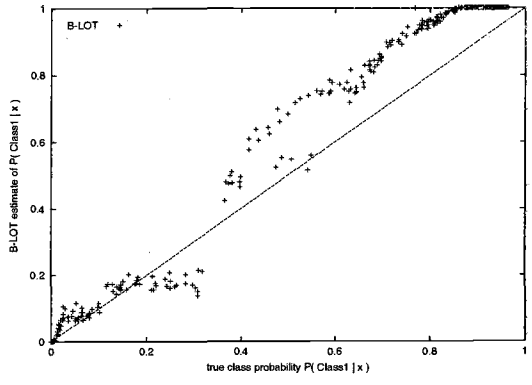
(a) B-PETs;TD=100;MINinst=2



(b) B-LOTs;TD=100;MINinst=2

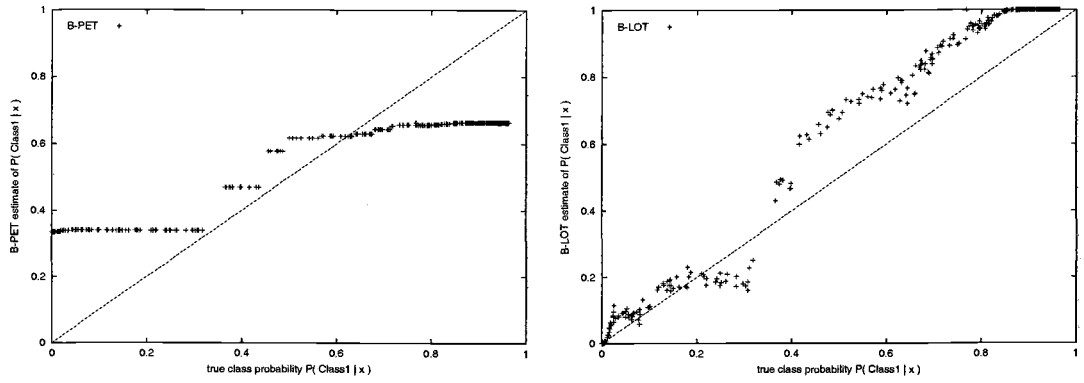


(c) B-PETs;TD=100;MINinst=8



(d) B-LOTs;TD=100;MINinst=8

FIGURE 6.26: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 100 instances), with the true class probabilities, for $MINinst = 2$ and $MINinst = 8$.



(a) B-PETs;TD=100;MINinst=12 (b) B-LOTs;TD=100;MINinst=12

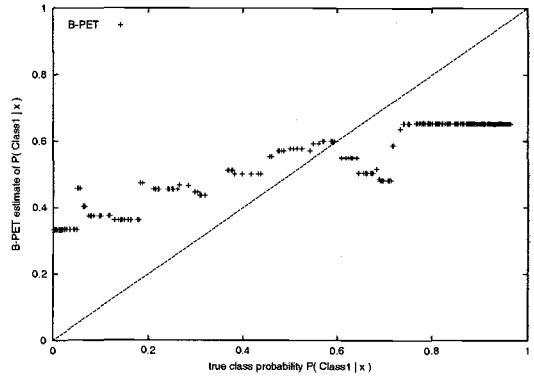
FIGURE 6.27: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 100 instances), with the true class probabilities, for $MINinst = 12$.

There are three fundamental mechanisms that combine into the B-LOTs for learning the class probability estimates: 1. Lazy learning 2. Options for splitting 3. Bagging

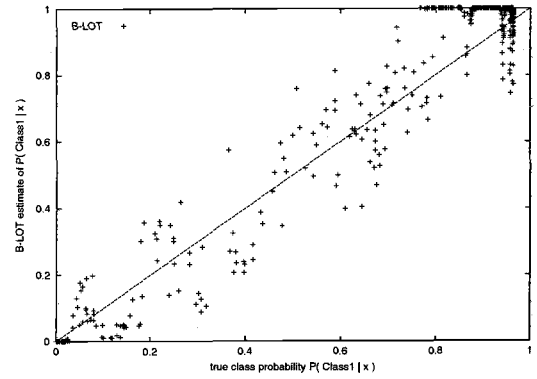
In order to have a better understanding of the reasons for which the B-LOTs perform well, we have removed each of the three mechanisms in turn from the algorithm and rerun the experiments.

Figures 6.37, 6.38, and 6.39 show the results of the tests. B-LT denotes the bagged lazy trees (no options), LOT denotes single lazy option trees (no Bagging), and LT denotes single lazy trees (no Bagging, no options).

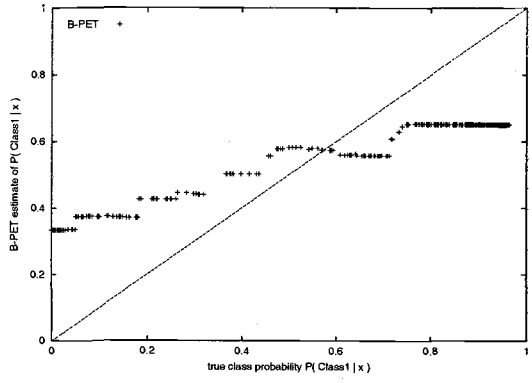
The lazy tree mechanism alone (i.e., the LTs, see Figure 6.39) scores more wins than the B-PETs only in the case of seven domains (audiology, iris, lymphography, segmentation, soybean, wine, and zoo). Figures Figures 6.37 and 6.38 show that both bagging and options help improve the performance of the lazy methods signif-



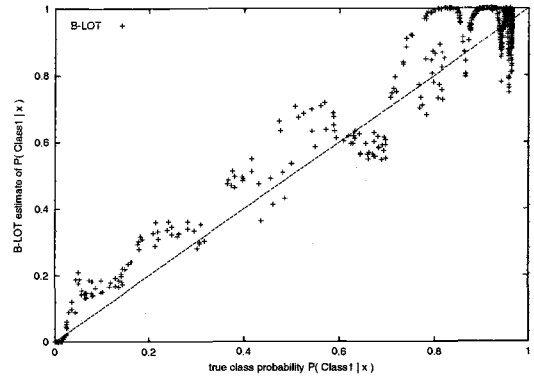
(a) B-PETs;TD=300;MINinst=2



(b) B-LOTs;TD=300;MINinst=2

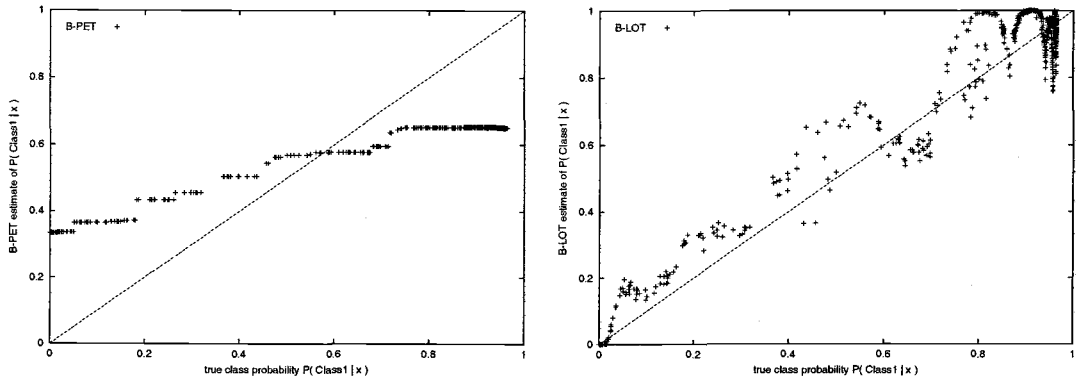


(c) B-PETs;TD=300;MINinst=8



(d) B-LOTs;TD=300;MINinst=8

FIGURE 6.28: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 300 instances), with the true class probabilities, for $MINinst = 2$ and $MINinst = 8$.



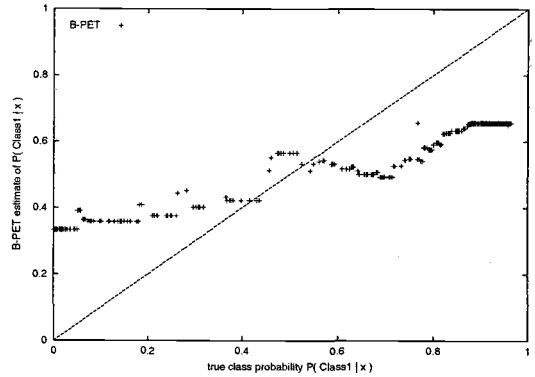
(a) B-PETs;TD=300;MINinst=12 (b) B-LOTs;TD=300;MINinst=12

FIGURE 6.29: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 300 instances), with the true class probabilities, for $MINinst = 12$.

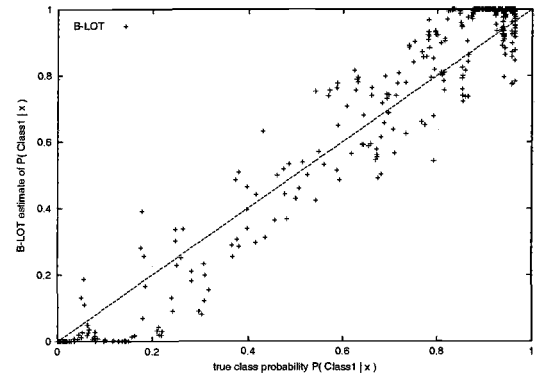
icantly. We can observe a slight advantage of the options mechanism over bagging, especially in the case of abalone, lung-cancer, and sonar. These experiments show that removing any of the two voting mechanisms (bagging and options) from the B-LOTs hurts the performance of the learned classifiers, suggesting that they might have complementary effects.

6.4.1 Weighting and Probability Estimation Methods Compared

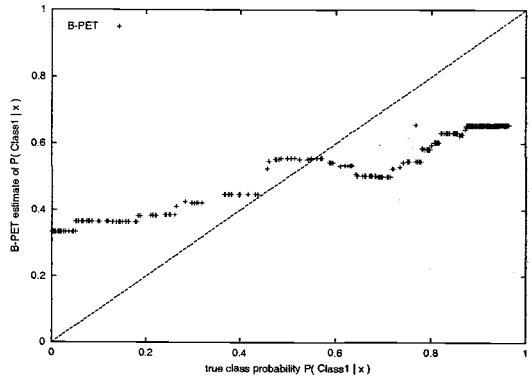
Both probability estimation methods (B-PETs and lazy tree based estimators) involve a much larger computational effort than the heuristics described in Chapter 3 and tested in Section 6.3.



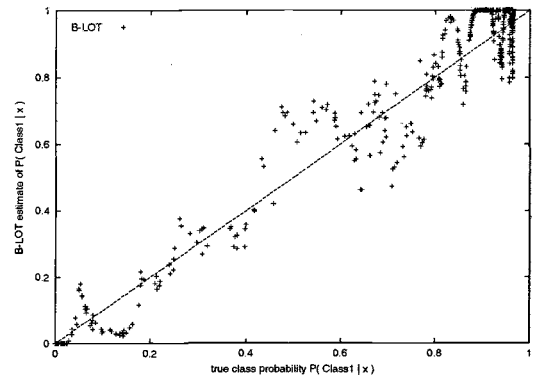
(a) B-PETs;TD=600;MINinst=2



(b) B-LOTs;TD=600;MINinst=2

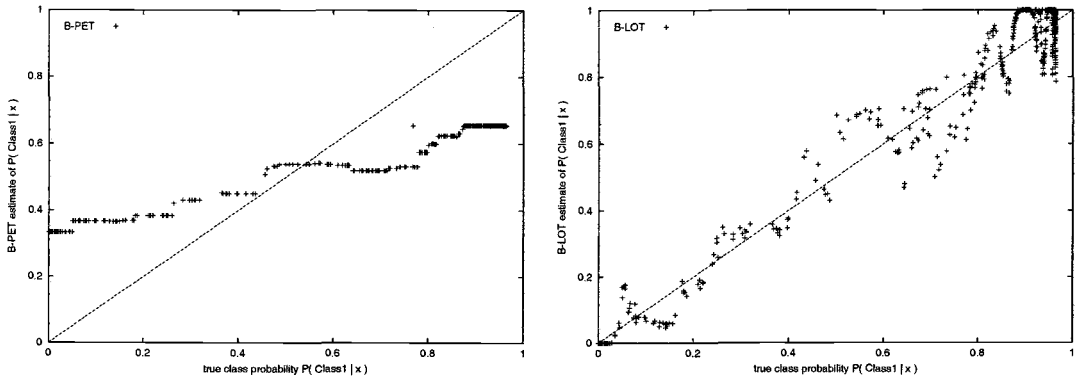


(c) B-PETs;TD=600;MINinst=8



(d) B-LOTs;TD=600;MINinst=8

FIGURE 6.30: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 600 instances), with the true class probabilities, for $MINinst = 2$ and $MINinst = 8$.



(a) B-PETs;TD=600;MINinst=12 (b) B-LOTs;TD=600;MINinst=12

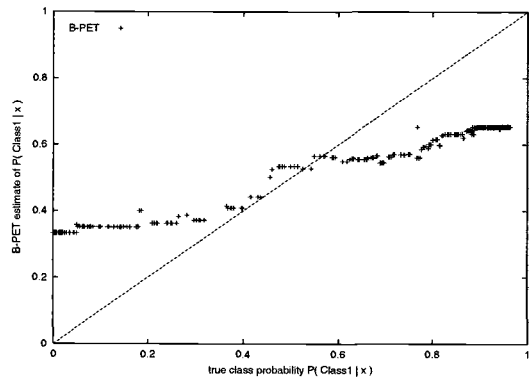
FIGURE 6.31: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 600 instances), with the true class probabilities, for $MINinst = 12$.

Therefore, we wanted to see whether this effort is worthwhile and compare the two best weighting methods against the B-LOTs and the B-PETs on the sisteen UCI domains. Figures 6.40, 6.41, 6.42, and 6.43 show the results.

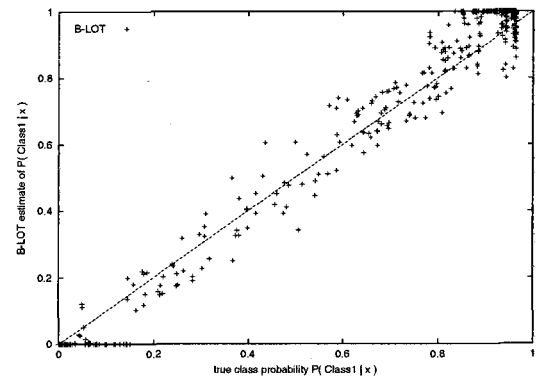
6.4.2 The River Channel Stability Task

We have also tested the bagged LOTs and the bagged PETs on data collected for a geology task: evaluating the stability of river channels in the Upper Colorado River Basin [128].

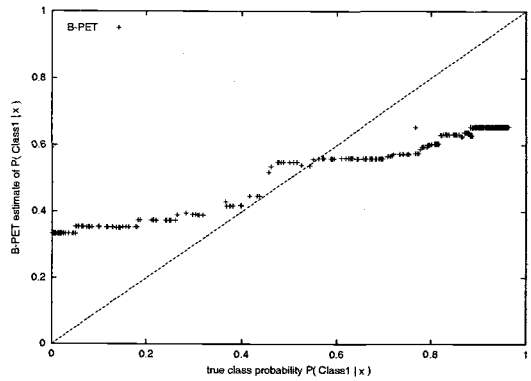
Over the course of the years, a river reaches a state of maximum efficiency in transporting sediment through its basin [20]. That state is usually associated with an energy equilibrium. More precisely, it is a dynamic equilibrium because rivers constantly scour and deposit sediments. This dynamic equilibrium is occasionally disrupted either because of human land use activities (e.g. mining, dams, urban



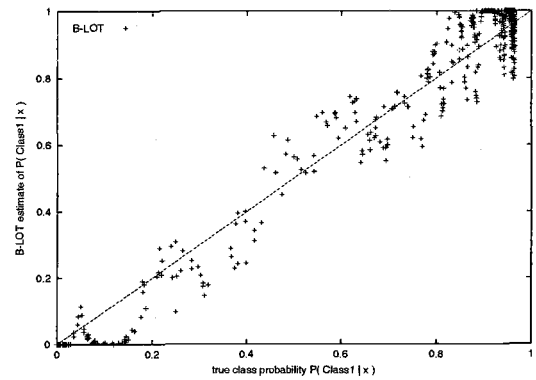
(a) B-PETs;TD=1000;MINinst=2



(b) B-LOTs;TD=1000;MINinst=2

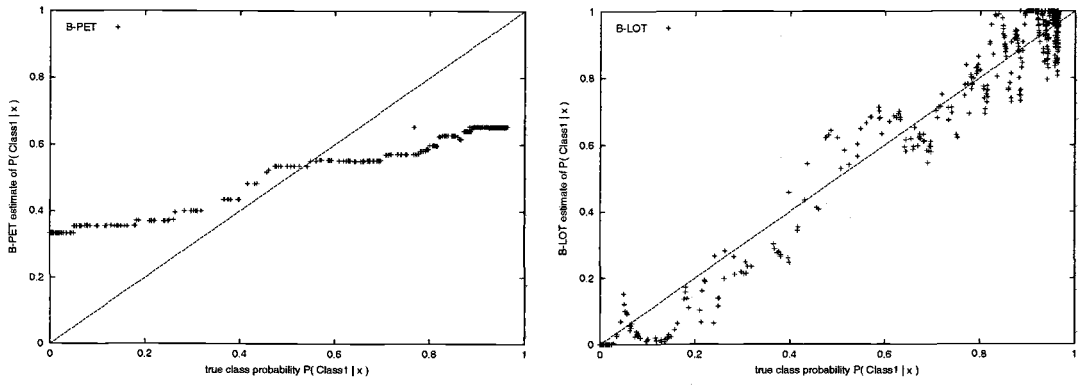


(c) B-PETs;TD=1000;MINinst=8



(d) B-LOTs;TD=1000;MINinst=8

FIGURE 6.32: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 1000 instances), with the true class probabilities, for $MINinst = 2$ and $MINinst = 8$.

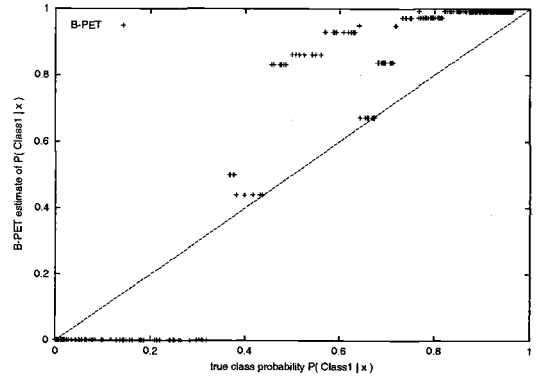


(a) B-PETs;TD=1000;MINinst=12 (b) B-LOTs;TD=1000;MINinst=12

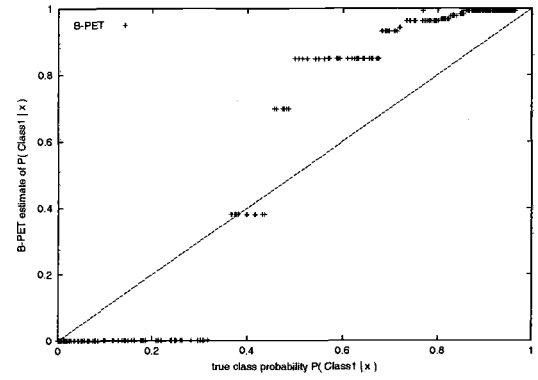
FIGURE 6.33: Scatter plots comparing the class probability estimates of the B-PETs and B-LOTs (trained on 1000 instances), with the true class probabilities, for $MINinst = 12$.

development), or as a result of natural disasters (e.g., glacial dam breakage), resulting in *unstable river channels.* Excessive channel instability can be detrimental especially when it has the potential of affecting human well being (e.g., it might cause landslides). Therefore, estimating river channel stability accurately is a highly important task.

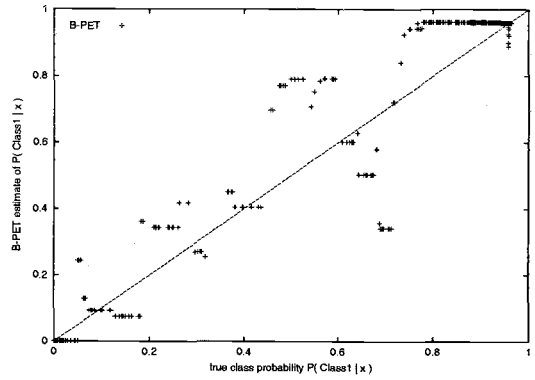
Stephanie Moret, Peter Klingeman and their group of researchers in the Environmental Sciences program at Oregon State University have collected channel stability data in fifty-five locations in the Upper Colorado River Basin, resulting in fifty-five examples. The data has eleven attributes: seven discrete and four continuous. In the original data, each example had a stability factor (a real-value between 55 and 117.5) associated with it, representing the output variable. Higher values of the stability factor mean higher instability of the corresponding channel. We have transformed the problem from its original regression format (the targets, represented by the sta-



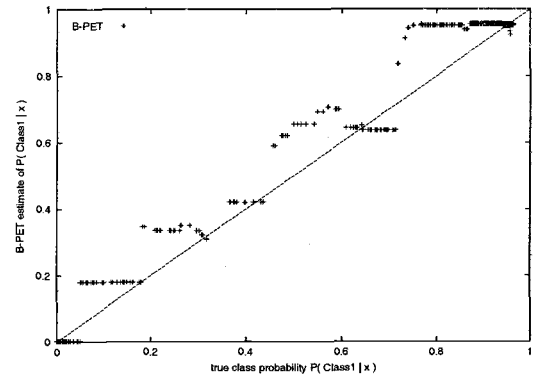
(a) B-PETs;TD=100;MINinst=2



(b) B-PETs;TD=100;MINinst=8

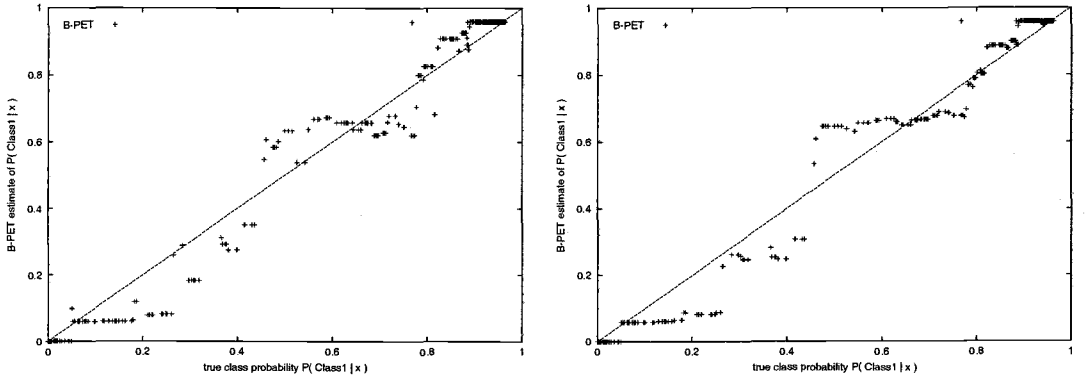


(c) B-PETs;TD=300;MINinst=2



(d) B-PETs;TD=300;MINinst=8

FIGURE 6.34: Scatter plots showing the class probability estimates of the B-PETs (with no Laplace correction) with the true class probabilities. $TD = 100$ and $TD = 300$.



(a) B-PETs;TD=1000;MINinst=2 (b) B-PETs;TD=1000;MINinst=8

FIGURE 6.35: Scatter plots showing the class probability estimates of the B-PETs (with no Laplace correction) with the true class probabilities. $TD = 1000$.

bility factors, were real valued numbers) into a two-class classification problem in which the classes are *stable* and *unstable*. The original stability values were computed using a table-based protocol (also called the Rapid Assessment Protocol, or RAP) provided by the US Department of Agriculture (USDA). The values in the table were calculated based on expert observations in the Rocky Mountain region. Therefore, we cannot assume that the values represent precise or reliable measurements, and furthermore the exact ("true") threshold θ_0 between stable and unstable stability factor values is unknown. To handle this situation, we have defined different classification problems by assigning the stable-unstable threshold θ different values from the [80,95] interval. These threshold values were suggested to us by the environmental sciences researchers studying this problem.

We first tested the algorithms in the absence of any cost matrix. To do this, we employed Leave-one-out cross-validation (LOOCV) [78, 99], because of the small size

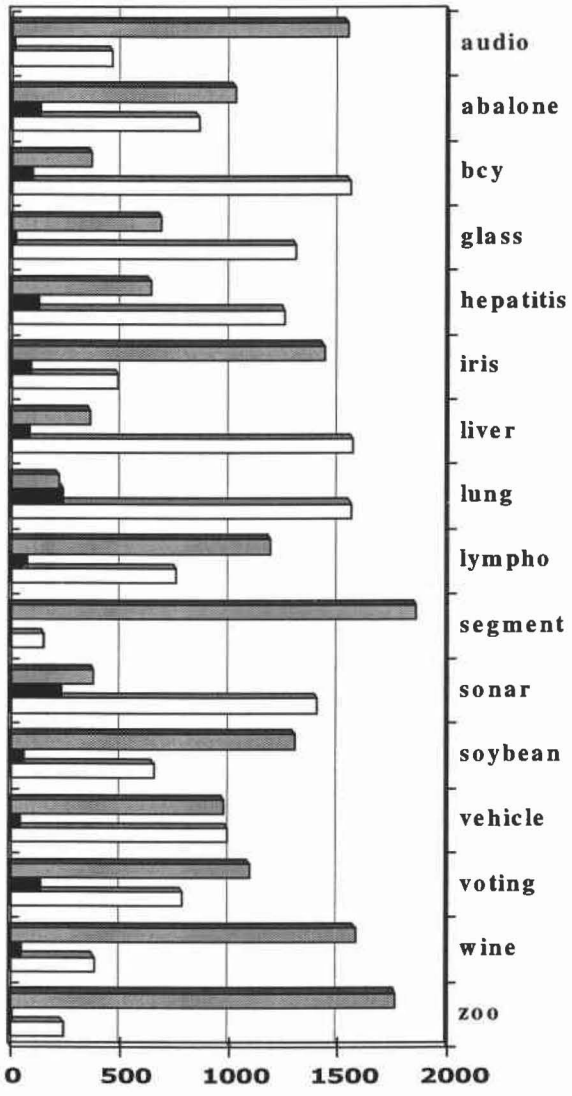


FIGURE 6.36: Barchart comparing the performance of B-LOTs and B-PETs for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-PETs, and the grey bars represent the number of wins of B-LOTs computed by BDELTACOST.

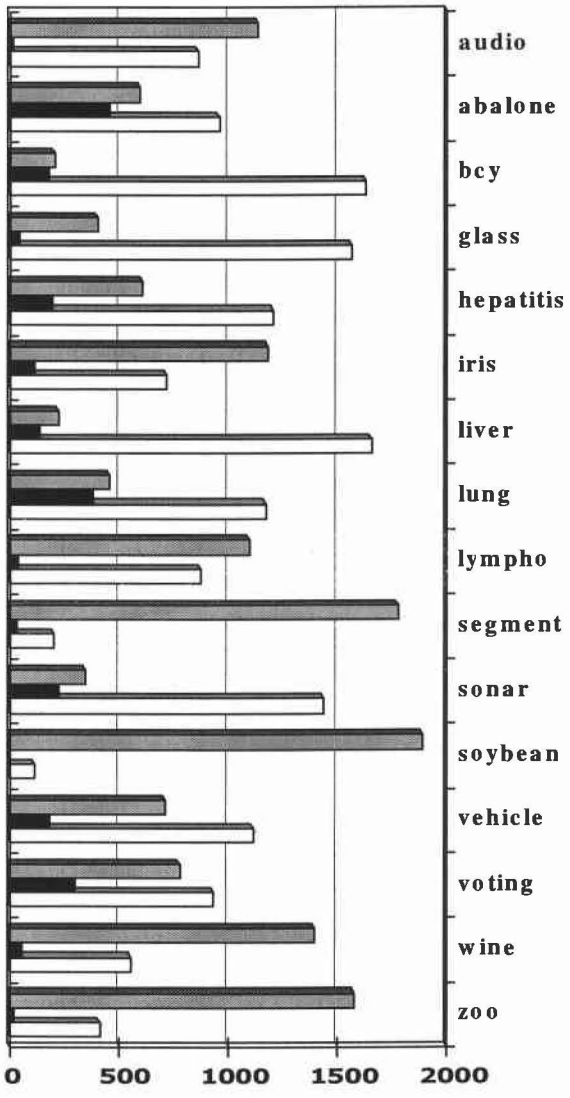


FIGURE 6.37: Barchart comparing the performance of the bagged lazy trees (B-LTs) and the B-PETs for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-PETs, and the grey bars represent the number of wins of B-LTs computed by BDELTACOST based on 95% confidence intervals.

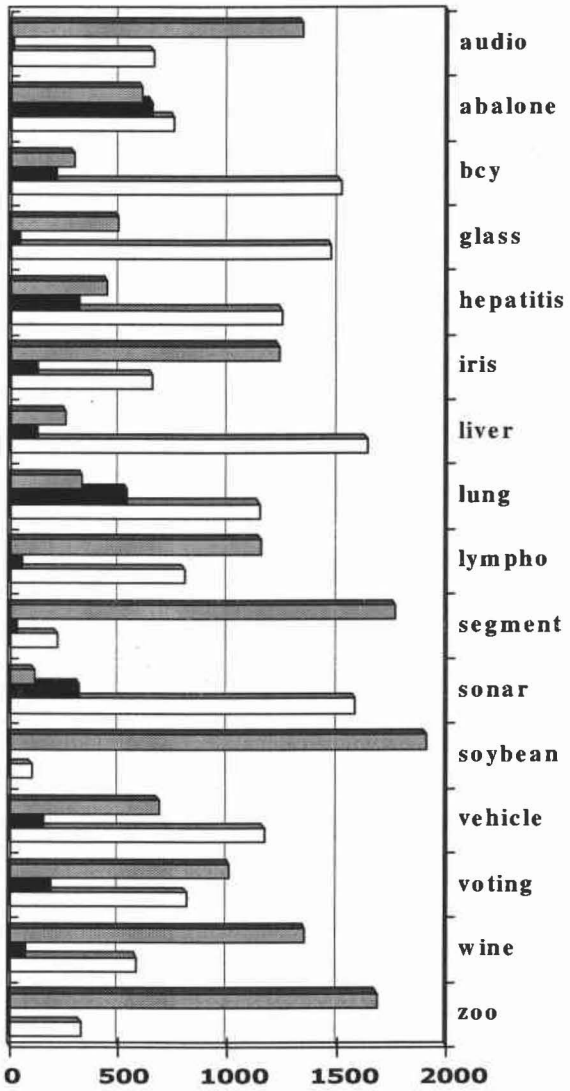


FIGURE 6.38: Barchart comparing the performance of the lazy option trees (LOTs) and the B-PETs for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-PETs, and the grey bars represent the number of wins of LOTs computed by BDeltaCost based on 95% confidence intervals.

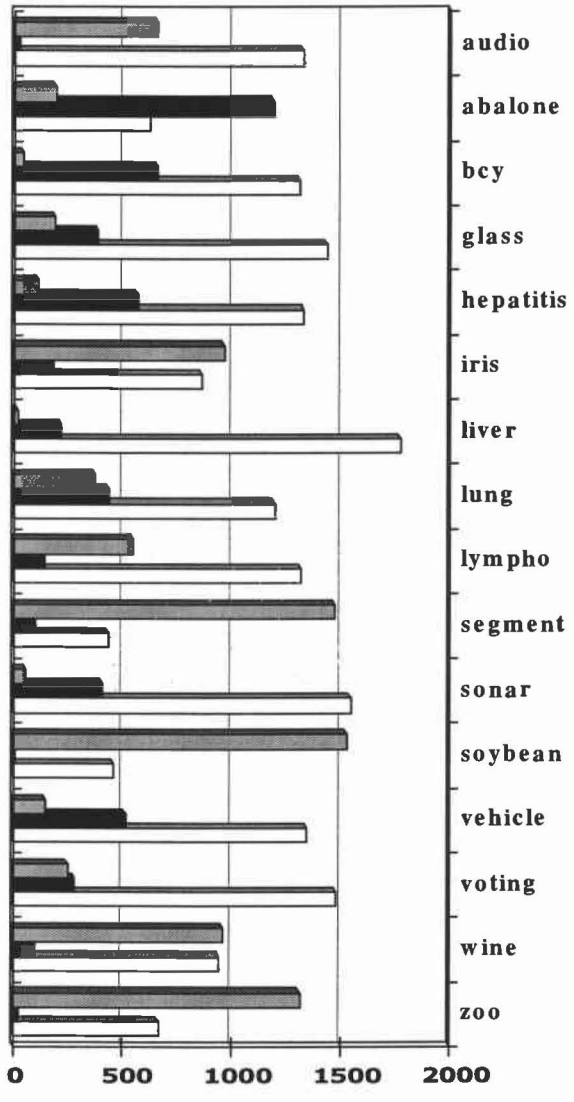


FIGURE 6.39: Barchart comparing the performance of the single lazy trees (LTs) and the B-PETs for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-PETs, and the grey bars represent the number of wins of LTs computed by BDELTACOST based on 95% confidence intervals.

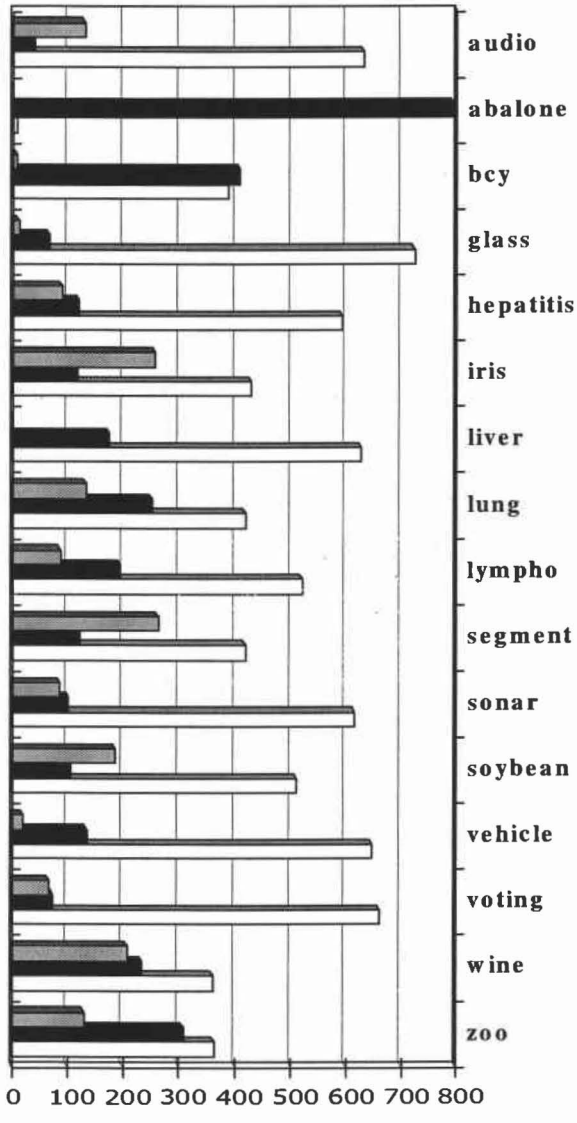


FIGURE 6.40: Barchart comparing the performance of the B-LOTs and *ClassFreq* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-LOTs, and the grey bars represent the number of wins of *ClassFreq* computed by BDELTACOST based on 95% confidence intervals.

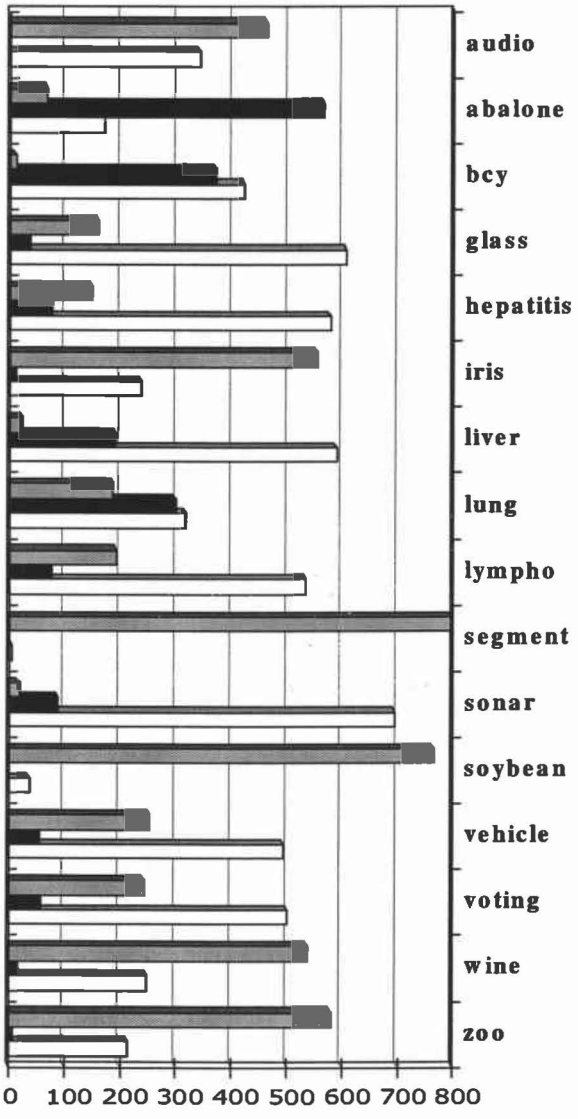


FIGURE 6.41: Barchart comparing the performance of the B-PETs and *ClassFreq* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-PETs, and the grey bars represent the number of wins of *ClassFreq* computed by BDELTACOST based on 95% confidence intervals.

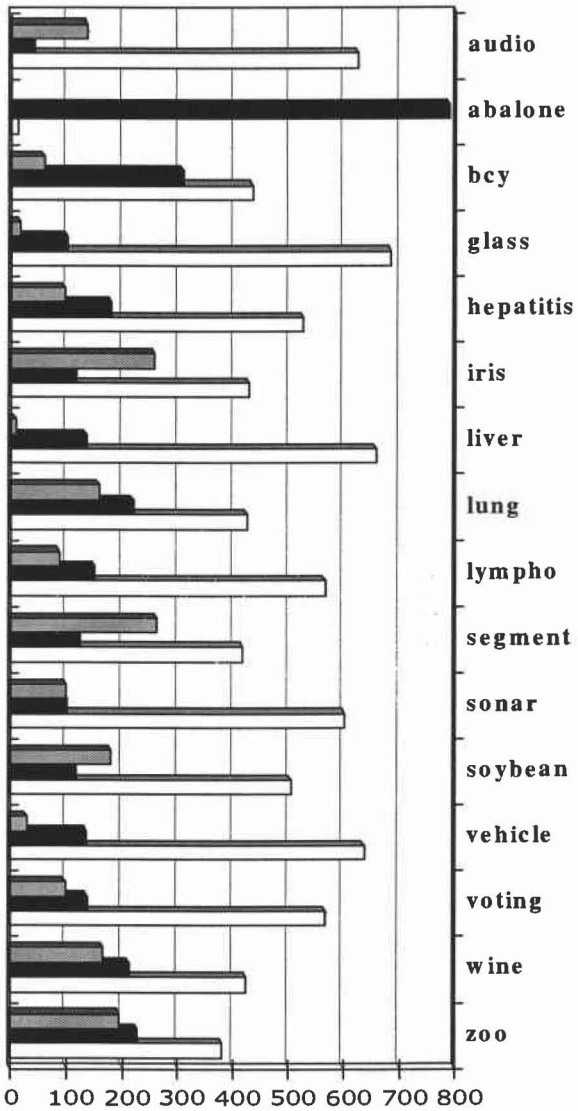


FIGURE 6.42: Barchart comparing the performance of the B-LOTs and *Powell20* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-LOTs, and the grey bars represent the number of wins of *Powell20* computed by BDELTACOST based on 95% confidence intervals.

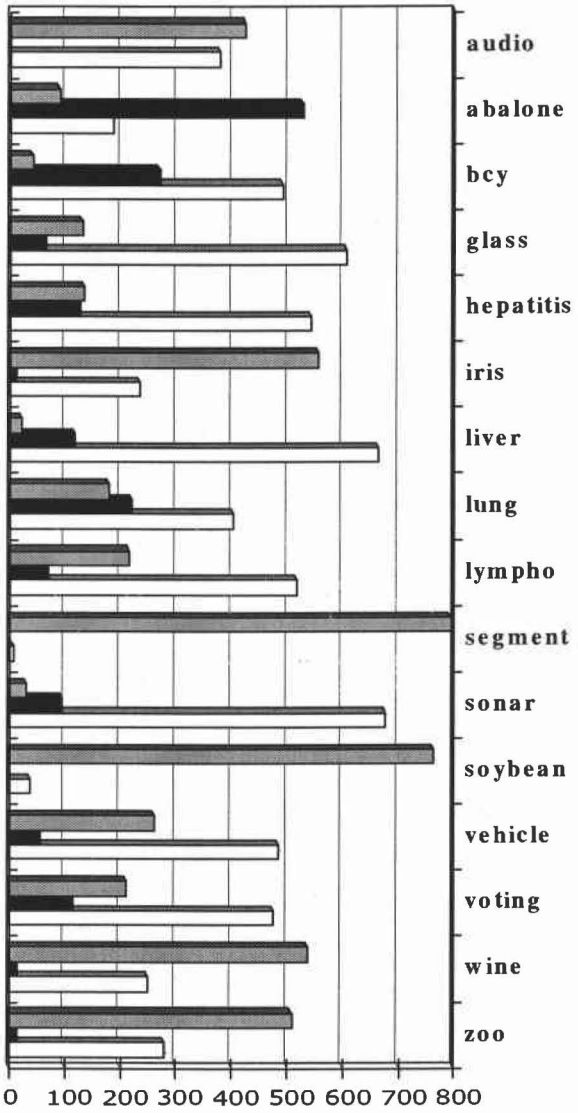


FIGURE 6.43: Barchart comparing the performance of the B-PETs and *Powell20* for the sixteen UCI data sets. White bars represent the number of ties, black bars represent the number of wins of B-PETs, and the grey bars represent the number of wins of *Powell20* computed by BDELTACOST based on 95% confidence intervals.

of the available data sample. Table 6.7 shows the values of the area under the ROC curve (AUC) for the two algorithms, for different values of the θ parameter.

The B-LOTs had better overall performance than the B-PETs for all values of θ except for the [90,93] range.

TABLE 6.7: Performance of the learning algorithms on the river channel stability task. This table shows the values of the Area Under the ROC Curve (AUC) for the two algorithms (B-LOTs and B-PETs) for different values of θ (the value that sets the threshold between stable and unstable river channels). All channels with a stability factor larger than θ are labeled as *unstable*, while all other channels are labeled as *stable*. These values were computed using leave-one-out cross-validation. The results printed with bold face indicate the values corresponding to the algorithm that was statistically significant better for a particular setting of θ.

θ	B-LOTs	B-PETs
80	**0.813**	0.743
81	0.839	0.814
84	0.818	0.795
86	**0.789**	0.743
88	**0.780**	0.652
90	0.673	0.684
93	0.610	**0.656**
95	0.671	0.652

Decisions when the Ranges in Costs are known.

The domain experts did not have precise values for the costs of the different decisions in the case of this problem. However, they were able to provide us a the range of values for the cost matrix. The ranges of the costs are represented in Table 6.8 (in US Dollars). The largest cost is the cost of the unstable channels that are classified as stable — a value in the order of tens of millions (representing the losses of human lives and losses of property that can be caused by an unstable channel that was not remediated). An incorrect classification of a stable channel has associated with it the cost of one day's work of an expert (sent to remediate the problem). A correct classification of an unstable channel will incur the cost of one day's expert work plus the cost of the remediation. The dominating cost however, is the cost of misclassifying an unstable river channel. Therefore the objectives of the learned classifiers are to classify all unstables correctly and to have as few misclassified stables as possible.

TABLE 6.8: The cost matrix for the river channel stability task, representing the ranges of costs associated with each decision.

	Correct Class	
Predicted Class	Stable	Unstable
Stable	0.0	human lives and property ($\sim 10^7$)
Unstable	one day ($\sim 10^3$)	one day + remediation ($\sim 10^3$)

For each threshold value θ, we have applied the bagged LOTs (B-LOTs) and the bagged PETs (B-PETs) to learn the ranking of the river channels in order to

make the decision that satisfies the two conditions described above. The results are summarized in Table 6.9.

TABLE 6.9: Performance of the learning algorithms on the river channel stability task. The values represent the proportion of stable channels that were misclassified (out of the total number of stable channels) when all unstable channels were classified correctly. θ is the value of the stability factor that sets the threshold between stable and unstable river channels. All channels with a stability factor that is larger than θ are labeled as *unstable* while all other channels are labeled as *stable*. Or, in other words, the table reports the false positive rate corresponding to a true positive rate of 1.0. A trivial classifier that would classify all examples as *unstable* has a proportion of misclassified *stables* of 1.0.

θ	B-LOTs	B-PETs
81	0.33	**0.22**
84	0.60	**0.40**
86	0.72	**0.45**
88	0.73	**0.55**
90	**0.47**	0.58
93	**0.50**	0.75
95	**0.28**	0.50

These results show that if the decision threshold is set optimally, the B-PETs give lower costs when $\theta < 90$ and the B-LOTs are better for larger values of θ. It is important to observe that larger values of θ correspond to fewer *unstable* examples in the data, creating a class imbalance.

This suggests that lazy learning was able to handle larger class imbalance better than the bagged PETs by focusing on the individual examples.

For all experiments that involved LOTs and B-LOTs in this chapter we have chosen the value for the maximum number of test allowed in a node to be $maxT = 3$ and the gain proportion $g = 0.2$. Preliminary experiments on some of the UCI data sets show that although the complexity of the trees grows a lot with the value of $maxT$, the results do not show major improvement of the LOTs or B-LOTs as $maxT$ is assigned larger values (we tested for $maxT = 5$, $maxT = 7$, and $maxT = 10$). In the meantime, larger values of g have proven to hurt the overall performance of the algorithm.

6.5 Summary

This chapter has presented the experimental results of testing the algorithms and methods presented in this dissertation.

First we validated the statistical tests introduced in Chapter 5. We have shown that these tests compute better confidence intervals than tests based on the normal distribution, and this makes them a useful tool for cost-sensitive testing of classifiers.

Next, we tested the data manipulation algorithms presented in Chapter 3. The results have shown that there is no clear winner among them but that *ClasFreq* can be preferred because it is inexpensive and gives a slightly better performance.

Finally, we have run experiments for testing the class probability estimation algorithms described in Chapter 4. The B-LOTs outperformed the B-PETs on the UCI domains. The bagging and options mechanisms together, have proven to improve significantly the performance of the algorithms. The probability estimates of the two algorithms in the case of one-dimensional overlapping Gaussians have shown that while the B-PETs give good rankings, the B-LOTs can sometimes give better

probability estimates, but these estimates leave a lot of room for improvement. We have also described the results of applying the probability estimation algorithms to the river channel task. The B-LOTs were able to classify more accurately the very expensive data points when these were underrepresented, showing that in the case of imbalanced data, lazy learning is a good option in practical applications.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Most of the practitioners in the areas of machine learning and data mining regard the problem of making predictions when the possible outcomes have different costs associated with them to be among the most important problems that machine learning research needs to address. As a result, a better understanding of the problem, and new and improved algorithms addressing this issue, are needed.

This dissertation has addressed that problem and presented methods that can be applied to solve it in a real-world setting.

This final chapter summarizes the main technical contributions of this thesis and discusses several promising directions for extending this research.

7.1 Contributions

The main goal of this dissertation was to give a structured overview of cost-sensitive learning and to develop new tools for cost-sensitive learning.

We have obtained the following results:

• We developed a new algorithm, the wrapper method based on Powell's optimization procedure for learning cost-sensitive classifiers by manipulating the training data. We also developed some simple heuristic methods for weighting. We have compared the wrapper method against the different heuristics. All these methods can be applied to problems with any number of classes while most existing algorithms based on the idea of stratification are applicable only to two-class problems. We found experimentally that *ClassFreq*, a heuristic that weights the examples based on

the original class distribution, and *Powell20*, the wrapper method using 20% of the data for validation outperform the other weighting methods.

- We developed a new class of algorithms for class probability estimation—the bagged Lazy Option Trees and their variants. In our experiments we compared the new methods against the bagged probability estimation trees. The B-LOTs have performed better than the B-PETs on the sixteen UCI data sets. A more detailed analysis has shown that both bagging and the options help improve the performance of the lazy methods. In the meantime, the B-PETs have computed very good rankings on a synthetic domain, but the probability estimates of the B-LOTs were slightly better.

- We introduced two new statistical tests for the cost-sensitive evaluation of classifiers. These are the first practical evaluation methods for cost-sensitive learning with a known cost matrix.

- We developed a new methodology for building the power curves to verify statistical tests for the comparison of two classifiers. We needed two classifiers that have the same cost but misclassify different examples. The second classifier was constructed by assigning the labels of the examples such that the two have the same confusion matrix. In order to generate the power curve, the labels of the second classifier were "damaged" randomly.

- We proposed a framework for categorizing and understanding cost-sensitive learning algorithms. This framework is based on the place (in the supervised learning process) in which the costs are processed.

7.2 Directions for Further Research

The research and the results described in this dissertation can be extended in many different directions.

7.2.1 Time and Space Dependent Cost Functions

This dissertation focuses on algorithms for cost functions represented by cost matrices. As mentioned in Chapter 2, many applications require cost functions that have spatial or temporal components. A subject for further research is the study of algorithms that handle such cost functions.

7.2.2 Lazy Class Probability Estimators

Chapter 4 has presented a class of lazy learning methods for class probability estimation. In many cases they performed better than the bagged PETs. However, if the practical application requires a very fast answer (e.g., an estimate for one million query points when the training data has millions of examples), the lazy algorithms will be much slower than the B-PETs. It would be of interest to study methods of speeding up the lazy class estimates for applications in which the new examples that need to be classified are grouped in batches. The experiments on the overlapping Gaussians have shown that there is a lot of room for improvement in both B-LOTs and B-PETs. Methods for calibrating rankings like the ones presented by Zadrozny and Elkan [191] could improve the probability estimates of these algorithms, given that they can produce good rankings.

7.2.3 Parametric Methods

The class probability estimation techniques described in Chapter 4 were applied to the cost-sensitive learning problem, but their applicability extends far beyond the scope of cost-sensitive learning. One major direction for future research is the study of the applicability of these non-parametric methods in areas in which parametric methods (like hidden Markov models) have been very successful (e.g., speech recognition).

7.2.4 Active Learning

Active learning is the field that studies learning algorithms in which data is acquired incrementally. In many real-world situations, there are high costs associated with collecting and labeling examples and obtaining or computing attribute values. Thus far, very little effort has been focused on using probability estimation for selecting new training examples or for selecting attributes to query. We have started applying the new probability estimators to these kinds of problems.

7.2.5 Hypothesis Testing

Cost-sensitive evaluation is not the only situation for which confidence intervals for non-Gaussian distributions (like the one described in Figure 5.1) are needed. Policy evaluation in the field of reinforcement learning, and any problem making use of data that has a multimodal distribution could also benefit from the two statistical tests described in Chapter 5.

7.3 Summary

Once a practical problem requiring a supervised learning approach is identified, it is important to first understand what error function needs to be minimized. In most cases, costs are distributed non-uniformly. Next, the user will have to select the algorithms that are appropriate for the task and the statistical tests that will be employed to assess the performance. This dissertation proposes a structured overview and new solutions for each of these steps for constructing learning systems that incorporate the costs of the different decisions.

BIBLIOGRAPHY

[1] Y. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6:192–198, 1990.

[2] N. M. Adams and D. J. Hand. Comparing classifiers when the misallocation costs are uncertain. *Pattern Recognition*, 32:1139–1147, 1999.

[3] A. Agresti. *Categorical Data Analysis.* John Wiley and Sons., Inc., 1990.

[4] D. Aha. A study of instance-based algorithms for supervised learning tasks. Ph.D. Thesis, Department of Information and Computer Sciences, University of California, Irvine, 1990.

[5] D. Aha. Generalizing from case studies: A case study. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 1–10. Morgan Kaufmann, 1992.

[6] D. Aha. Special issue on lazy learning. *Artificial Intelligence Review*, 1997.

[7] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[8] K. M. Ali. Learning probabilistic relational concept descriptions. Ph.D. Thesis, Department of Information and Computer Sciences, University of California, Irvine, 1996.

[9] E. Alpaydin. Combined 5x2 CV-F test for comparing supervised classification learning algorithms. *Neural Computation*, 8(11):1885–1892, 1999.

[10] D. Applebaum. *Probability and Information: An Integrated Approach.* Cambridge University Press, Cambridge, UK, 1996.

[11] P. Baldi and S. Brunak. *Bioinformatics - The Machine learning Approach.* MIT Press, second edition, 2001.

[12] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1/2), 1999.

[13] A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification with gene expression profiles. In R. Shamir, S. Miyano, S. Istrail, O. Pevzner, and M. Waterman, editors, *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, pages 54–64. ACM Press, 2000.

[14] K. P. Bennett and C. Campbell. Support vector machines: Hype or hallelujah. *SIGKDD Explorations*, 2(2), 2001.

[15] I. Bhandari, E. Colet, J. Parker, Z. Pines, and R. Pratap. Advanced scout: Data mining and knowledge discovery in NBA data. *Data Mining and Knowledge Discovery*, 1(1), 1997.

[16] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[17] Y. M. M. Bishop, S. E. Fienberg, and P. W. Holland. *Discrete multivariate analysis: Theory and practice.* MIT Press, Cambridge, Mass, 1975.

[18] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences, 1998. [`http://www.ics.uci.edu/~mlearn/MLRepository.html`].

[19] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[20] P. R. Bonneau and R. S. Snow. Character headwaters adjustment to base level drop, investigated by digital modeling. *Geomorphology*, 5:475–487, 1992.

[21] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[22] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building.* J. Wiley and Sons., Inc., 1978.

[23] J. Boyan, D. Freitag, and T. Joachims. A machine learning architecture for optimizing web search engines. In *Proceedings of the AAAI Workshop on Internet-Based Information Systems*, 1996.

[24] J. P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley. Pruning decision trees with misclassification costs. In C. Nedellec and C. Rouveirol, editors, *Lecture Notes in Artificial Intelligence. Machine Learning: ECML-98, Tenth European Conference on Machine Learning, Proceedings*, volume 1398, pages 131–136, Berlin, New York, 1998. Springer Verlag.

[25] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

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

[27] L. Breiman. Bias, variance, and arcing classifiers. Technical report, Department of Statistics, University of California, Berkeley, 1996.

[28] L. Breiman. Arcing classifiers. Technical report, Department of Statistics, University of California, Berkeley, 1997.

[29] L. Breiman. Out-of-bag estimation. Technical report, Department of Statistics, University of California, Berkeley, 1998.

[30] L. Breiman. The mysteries of prediction. In *Keynote Address to the 31st Symposium on the Interface*, 1999.

[31] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[32] C. E. Brodley and P. Smyth. The process of applying machine learning algorithms. In *Working Notes for Applying Machine Learning in Practice: A Workshop at the Twelfth International Conference on Machine Learning*, pages 7–13. NRL, 1995.

[33] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

[34] W. Buntine. A theory of learning classification rules. Ph.D. Thesis, School of Computing Science, University of Technology, Sydney, Australia, 1990.

[35] W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence Frontiers in Statistics*, pages 182–201, London, UK, 1993. Chapman & Hall.

[36] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85, 1992.

[37] M. C. Burl, L. Asker, P. Smyth, U. M. Fayyad, P. Perona, L. Crumpler, and J. Aubele. Learning to recognize volcanoes on Venus. *Machine Learning*, 30(2/3):165–194, 1998.

[38] I. V. Cadez, P. Smyth, G. J. McLachlan, and C. E. McLaren. Maximum likelihood estimation of mixture densities for binned and truncated multivariate data. *Machine Learning, Special Edition on Unsupervised Learning, to appear*, 2001.

[39] B. Cestnik. Estimating probabilities: A crucial task in machine learning. In L. C. Aiello, editor, *Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 147–149, London, 1990. Pitman Publishing.

[40] K. J. Cherkauer. Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In P. Chan, editor, *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, pages 15–21, 1996.

[41] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

[42] P. R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, Cambridge, MA, 1995.

[43] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* J.Wiley and Sons, Inc., 1991.

[44] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[45] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner. Gradient based learning applied to document recognition. *Transactions of the IEEE*, 86(11):2278–2324, 1998.

[46] J. Cussens and S. Dzeroski. *Learning Language in Logic.* Springer Verlag, 2000.

[47] B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA, 1990.

[48] B. V. Dasarathy and L. J. White. A characterization of nearest neighbor decision surfaces and a new approach to generate them. *Pattern Recognition*, 10:41–46, 1978.

[49] G. R. Dattatreya and V. V. S. Sarma. Bayesian and decision tree approaches for pattern recognition including feature measurement costs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(3):293–298, 1981.

[50] M. H. DeGroot. *Probability and Statistics*. Addison-Wesley Publishing Company, Reading, Mass., 1975.

[51] T. G. Dietterich. Machine-learning research: Four current directions. *AI Magazine*, 18(4), 1997.

[52] T. G. Dietterich. Statistical tests for comparing supervised classification. *Neural Computation*, 10:1895–1924, 1998.

[53] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. *Machine Learning*, 40(2):139–158, 2000.

[54] T. G. Dietterich and G. Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 572–577. AAAI Press/MIT Press, 1991.

[55] T. G. Dietterich, H. Hild, and G. Bakiri. A comparison of ID3 and Backpropagation for English text-to-speech mapping. *Machine Learning*, 18:51–80, 1995.

[56] T. G. Dietterich, M. Kearns, and Y. Mansour. Applying the weak learning framework to understand and improve C4.5. In Lorenza Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 96–104, San Francisco, CA, 1996. Morgan Kaufmann.

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

[58] A. Dobra and J. Gehrke. Bias correction in classification tree construction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 90–97, San Francisco, CA, 2001. Morgan Kaufmann.

[59] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 155–164, New York, 1999. ACM Press.

[60] C. Drummond and R. C. Holte. Explicitly representing expected cost: An alternative to ROC representation. In *ICML-2000 Workshop Notes Cost-Sensitive Learning*, pages 37–46, 2000.

[61] C. Drummond and R. C. Holte. Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *Machine Learning: Proceedings of the Seventeenth International Conference*, pages 239–246, San Francisco, CA, 2000. Morgan Kaufmann.

[62] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, Inc., 1973.

[63] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* John Wiley and Sons, Inc. - Interscience, second edition, 2000.

[64] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* Chapman and Hall, New York, 1993.

[65] J. P. Egan. *Signal detection theory and ROC-analysis.* Academic Press, New York, NY, 1975.

[66] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., 2001.

[67] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: Misclassification cost-sensitive boosting. In *Machine Learning: Proceedings of the Sixteenth International Conference*, pages 97–105, San Francisco, 1999. Morgan Kaufmann.

[68] T. Fawcett and F. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3), 1997.

[69] T. Fawcett and F. Provost. Activity monitoring: Noticing interesting changes in behavior. In S. Chaudhuri and D. Madigan, editors, *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999.

[70] U. M. Fayyad, N. Weir, and S. Djorgovski. SKICAT: A machine learning system for automated cataloging of large scale sky surveys. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 112–119. Morgan Kaufmann, 1993.

[71] S. E. Fienberg and P. W. Holland. Methods for eliminating zero counts in contingency tables. In G. P. Patil, editor, *Random Counts in Scientific Work: Volume 1.* The Pennsylvania State University Press, 1968.

[72] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, pages 256–285, 1995.

[73] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Lecture Notes in Artificial Intelligence, Proceedings ECML-95.* Springer Verlag, 1995.

[74] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 1997.

[75] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the International Conference in Machine Learning*, pages 148–156, San Francisco, CA, 1996. Morgan Kaufmann.

[76] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Stanford University, Department of Statistics, 1998.

[77] J. H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 717–724. AAAI Press/MIT Press, 1996.

[78] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, New York, NY, 1972.

[79] M. R. Garey and D. S. Johnson. *Computers and Intractability, a guide to the Theory of NP-Completeness.* W. H. Freeman and Co., New York, 1978.

[80] I. J. Good. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods.* M.I.T. Press, Cambridge, Mass., 1965.

[81] I. J. Good. *Good thinking: The foundations of probability and its applications.* University of Minnesota Press, Minneapolis, MN, 1983.

[82] D. J. Hand. *Construction and Assessment of Classification Rules.* John Wiley and Sons.,Inc., 1996.

[83] D. J. Hand and R. J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. To appear. *Machine Learning*, 2001.

[84] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

[85] J. A. Hanley and B. J. McNeil. A method of comparing the areas under the receiver operating characteristic curves derived from the same cases. *Radiology*, 148:839–843, 1983.

[86] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12:993–1001, 1990.

[87] P. E. Hart. The condensed nearest-neighbor rule. *IEEE Transactions on Information Theory*, IT-4:515–516, 1968.

[88] T. Hastie and R. Tibshirani. Classification by pairwise coupling. Technical report, Department of Statistics, Stanford University, 1996.

[89] D. Haverkamp, C. Tsatsoulis, and S. Gogineni. The combination of algorithmic and heuristic methods for the classification of sea ice imagery. *Remote Sensing Reviews*, 9:135–159, 1994.

[90] S. Haykin. *Neural Networks. A Comprehensive Foundation.* Macmillan Publishing Company, 1994.

[91] D. Heckerman, D.M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for density estimation, collaborative filtering, and data visualization. In *Proceedings of Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 82–88, San Francisco, CA, 2000. Morgan Kaufmann.

[92] A. Ittner and M. Schlosser. Non-linear decision trees. In *Proceedings of the Thirteenth International Conference on Machine Learning*, San Francisco, CA, 1996. Morgan Kaufmann.

[93] A. C. Jain, R. C. Dubes, and C.-C. Chen. Bootstrap techniques for error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):628–633, 1987.

[94] N. Japkowicz. The class imbalance problem: Significance and strategies. In *Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000)*, 2000.

[95] F. Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MA, 1997.

[96] F. V. Jensen. *An Introduction to Bayesian Networks.* Springer Verlag, 1996.

[97] R. M. Karp. Reducibility among combinatorial problems. In *IBM Symposium on Computational Complexity*, pages 85–103, 1973.

[98] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 459–468. ACM Press, 1996.

[99] M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11(6):1427–1453, 1999.

[100] U. Knoll, G. Nakhaeizadeh, and B. Tausend. Cost-sensitive pruning of decision trees. In F. Bergadano and L. DeRaedt, editors, *Lecture Notes in Artificial Intelligence. Machine Learning: ECML-94, European Conference on Machine Learning, Proceedings*, volume 784, pages 383–386, Berlin, New York, 1994. Springer Verlag.

[101] R. Kohavi and C. Kunz. Option decision trees with majority votes. In D. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 161–169, San Francisco, CA, 1997. Morgan Kaufmann.

[102] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 170–178. Morgan Kaufmann, 1997.

[103] E. B. Kong and T. G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proceedings of the International Conference in Machine Learning*, pages 313–321, San Francisco, CA, 1995. Morgan Kaufmann.

[104] E. B. Kong and T. G. Dietterich. Probability estimation via error-correcting output coding. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, Banff, Canada, 1997.

[105] M. Kubat, R. Holte, and S. Matwin. Learning when negative examples abound. In *Lecture Notes in Artificial Intelligence, Proceedings of ECML-97*, volume 1224, pages 146–153. Springer Verlag, 1997.

[106] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2/3), 1998.

[107] M. Kubat and S. Matwin. Addressing the curse of imbalanced training sets: One-sided sampling. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[108] M. Kukar and I. Kononenko. Cost-sensitive learning with neural networks. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, Chichester, NY, 1998. Wiley.

[109] J. Laird and M. van Lent. Interactive computer games: Human-level AI's killer application. *AI Magazine*, 22(2), 2001.

[110] P. Latinne, M. Saerens, and C. Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities may significantly improve classification accuracy: Evidence from a multi-class problem in remote sensing. In *Proceedings of the Eighteenth International Conference on Machine Learning*, San Francisco, CA, 2001. Morgan Kaufmann.

[111] N. Lavrac, D. Gamberger, and P. Turney. Cost-sensitive feature reduction applied to a hybrid genetic algorithm. In *Proceedings of the Seventh International Workshop on Algorithmic Learning Theory, ALT'96*, pages 127–134, 1996.

[112] S. Lawrence, I. Burns, A. Black, A. C. Tsoi, and L. Giles. Neural network classification and prior class probabilities. In G. Orr, K. R. Muller, and R. Caruana, editors, *Tricks of the Trade, Lecture Notes in Computer Science State-of-the-Art Surveys*, pages 299–314. Springer Verlag, 1998.

[113] W. Lee. A data mining framework for constructing features and models for intrusion detection systems. Ph.D. Thesis, Department of Computer Science, Columbia University, New York, NY, 1999.

[114] Y. Lee, B. G. Buchanan, and J. M. Aronis. Learning rules to predict rodent carcinogenicity. *Machine Learning*, 30(2/3):217–240, 1998.

[115] R. Liere and P. Tadepalli. Active learning with committees for text categorization. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, pages 591–596, 1997.

[116] T. S. Lim, W. Y. Loh, and Y. S. Shih. A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000.

[117] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

[118] D. D. Margineantu. Building ensembles of classifiers for loss minimization. In M. Pourahmadi, editor, *Models, Predictions and Computing: Proceedings of the 31st Symposium on the Interface*, volume 31, pages 190–194. The Interface Foundation of North America, 1999.

[119] D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 211–218. Morgan Kaufmann, 1997.

[120] D. D. Margineantu and T. G. Dietterich. Learning decision trees for loss minimization in multi-class problems. Technical Report 99-30-03, Department of Computer Science, Oregon State University, 1999.

[121] D. D. Margineantu and T. G. Dietterich. Bootstrap methods for the cost-sensitive evaluation of classifiers. In *Machine Learning: Proceedings of the Seventeenth International Conference*, pages 583–590, San Francisco, CA, 2000. Morgan Kaufmann.

[122] D. D. Margineantu and T. G. Dietterich. Lazy class probability estimators. In *Proceedings of the 33rd Symposium on the Interface*, 2001.

[123] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161, 1983.

[124] R. S. Michalski. *Machine Learning: A Multistrategy Approach*. Morgan Kaufmann, San Mateo, CA, 1994.

[125] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning: An Artificial Intelligence Approach.* Tioga, Palo Alto, CA, 1983.

[126] J. Mingers. Expert systems - rule induction with statistical data. *Journal of the Operational Research Society*, 38:39–47, 1987.

[127] T. Mitchell. *Machine Learning.* The McGraw-Hill Companies, Inc., 1997.

[128] S. Moret. Predicting channel stability in Colorado mountain streams using hydrobiogeomorphic and land use data: A cost-sensitive machine learning approach to modeling rapid assessment protocols. Ph.D. Thesis, Environmental Sciences, Oregon State University, Corvallis, OR, 2001.

[129] S. Moret, D. D. Margineantu, W. Langford, and D. Holdt. Predicting rapid assessment protocols using machine learning techniques: Channel stability. In *Proceedings of the 2001 American Water Resources Association Conference on Decision Support Systems for Water Resources Management*, 2001.

[130] D. F. Morrison. *Multivariate Statistical Methods, Third edition.* McGraw-Hill, Englewood Cliffs, NJ, 1990.

[131] S. H. Muggleton, C. H. Bryant, and A. Srinivasan. Measuring performance when positives are rare: Relative advantage versus predictive accuracy - a biological case-study. In R. L. de Mantaras and E. Plaza, editors, *Proceedings of the Eleventh European Conference on Machine Learning, Lecture Notes in Artificial Intelligence.* Springer Verlag, 2000.

[132] S. Murthy, S. Kasif, and S. Salzberg. Randomized induction of oblique decision trees. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, Cambridge, MA, 1993. MIT Press.

[133] D. W. Opitz and J. W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Advances in Neural Information Processing Systems, 8*, pages 535–541, Cambridge, MA, 1996. MIT Press.

[134] G. Pagallo and D. Haussler. Two algorithms that learn DNF by discovering relevant features. In *Proceedings of the Sixth International Machine Learning Workshop.* Morgan Kaufmann, 1989.

[135] B. Parmanto, P. W. Munro, and H. R. Doyle. Improving committee diagnosis with resampling techniques. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8. Lawrence Earlbaum Associates, 1996.

[136] B. Parmanto, P. W. Munro, and H. R. Doyle. Reducing variance of committee prediction with resampling techniques. *Connection Science*, 8(3–4):405–426, 1996.

[137] G. P. Patil, editor. *Random Counts in Scientific Work: Volume 1.* The Pennsylvania State University Press, 1968.

[138] T. R. Payne. Learning email filtering rules with Magi. Ph.D. Thesis, Department of Computing Science, University of Aberdeen, Abredeen, Scotland, 1994.

[139] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk. Reducing misclassification costs. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 217–225. Morgan Kaufmann, 1994.

[140] J. H. Piater, P. R. Cohen, X. Zhang, and M. Atighetchi. A randomized ANOVA procedure for comparing performance curves. In *Machine Learning: Proceedings of the Fifteenth International Conference.* Morgan Kaufmann, 1998.

[141] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 1999.

[142] T. Poggio and F. Girosi. Networks for approximation and learning. In *Proceedings of the IEEE, Special Issue on Neural Networks: Theory and Modeling*, volume 78(9), pages 1481–1497, 1990.

[143] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C - The Art of Scientific Computing, 2nd ed.* Cambridge University Press, 1992.

[144] F. Provost and P. Domingos. Well-trained PETs: Improving probability estimation trees. Technical Report IS-00-04, Stern School of Business, New York University, 2000.

[145] F. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press, 1997.

[146] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–232, 2001.

[147] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing classifiers. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, San Francisco, 1998.

[148] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[149] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Francisco, 1993.

[150] J. R. Quinlan. Bagging, Boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, Cambridge, MA, 1996. AAAI Press/MIT Press.

[151] G. C. Reinsel. *Elements of Multivariate Time Series Analysis.* Springer Verlag, 1993.

[152] M. D. Richard and R. P. Lippman. Neural network classifiers estimate bayesian *a posteriori* probabilities. *Neural Computation*, 3(4):461–483, 1991.

[153] J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, IT-30(4):629–636, 1984.

[154] B. E. Rosen. Ensemble learning using decorrelated neural networks. *Connection Science*, 8(3–4):373–384, 1996.

[155] D. B. Rosen. How good were those probability predictions? The expected recommendation loss (ERL scoring rule). In G. Heidbreder, editor, *Maximum Entropy and Bayesian Methods. Proceedings of the Thirteenth International Workshop.*, Doordrecht, The Netherlands, 1995. Kluwer Academic Publishers.

[156] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McCleland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.

[157] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ, 1996.

[158] S. Salzberg, R. Chandar, H. Ford, S. Murthy, and R. White. Decision trees for automated identification of cosmic ray hits in Hubble space telescope images. *Publications of the Astronomical Society of the Pacific*, 107:1–10, 1995.

[159] S. Salzberg, A. Delcher, K. Fasman, and J. Henderson. A decision tree system for finding genes in DNA. *Journal of Computational Biology*, 5(4):667–680, 1998.

[160] R. E. Schapire. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 313–321. Morgan Kaufmann, 1997.

[161] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the efffectiveness of voting methods. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 322–330. Morgan Kaufmann, 1997.

[162] N. Shang and L. Breiman. Distribution based trees are more accurate. Technical report, University of California, Berkeley, 1996.

[163] R. E. Shapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[164] J. Shavlik and T. G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1990.

[165] P. Smyth, A. Gray, and U. Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In *Machine Learning: Proceedings of the Twelvth International Conference*, pages 506–514, 1995.

[166] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Predictiona and Search.* MIT Press, second edition, 2001.

[167] A. Srinivasan. Note on the location of optimal classifiers in n-dimensional ROC space. Technical Report PRG-TR-2-99, Oxford University Computing Laboratory, Oxford, UK, 1999.

[168] W. N. Street. Cancer diagnosis and prognosis via linear-programming-based machine learning. Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, WI, 1994.

[169] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems.* V. H. Winston and Sons, J. Wiley and Sons, Washington, D.C., 1977.

[170] K. M. Ting and Z. Zheng. Boosting trees for cost-sensitive learning. In C. Nedellec and C. Rouveirol, editors, *Lecture Notes in Artificial Intelligence. Machine Learning: ECML-98, Tenth European Conference on Machine Learning, Proceedings*, volume 1398, pages 190–195, Berlin, New York, 1998. Springer Verlag.

[171] R. Tjoelker and W. Zhang. A general paradigm for applying machine learning in automated manufacturing processes. In *Conference on Automated Learning and Discovery, CONALD'98. Workshop on Reinforcement Learning and Machine Learning for Manufacturing*, 1998.

[172] G. T. Toussaint, B. K. Bhattacharya, and R. S. Poulsen. The application of Voronoi diagrams to non-parametric decision rules. In *Proceedings of the 16th Symposium on Computer Science and Statistics: The Interface*, pages 97–108, 1987.

[173] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3–4):385–404, 1996.

[174] P. Turney. Exploiting context when learning to classify. In *Proceedings of the European Conference on Machine Learning*, pages 402–407. Springer Verlag, 1993.

[175] P. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995.

[176] P. Turney. Data engineering for the analysis of semiconductor manufacturing data. In *Proceedings of the IJCAI-95 Workshop on Data Engineering for Indctive Learning*, pages 50–59, 1995.

[177] P. Turney. Cost-sensitive learning bibliography. Online Bibliography. Institute for Information Technology of the National Research Council of Canada, Ottawa., 1997. [`http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html`].

[178] P. Turney. Types of cost in inductive concept learning. In *ICML-2000 Workshop Notes Cost-Sensitive Learning*, pages 15–21, 2000.

[179] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[180] P. E. Utgoff and C. E. Brodley. Linear machine decision trees. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 1991.

[181] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, 1995.

[182] V. N. Vapnik. *Statistical Learning Theory.* John Wiley and Sons., Inc., 1998.

[183] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, second edition, 2000.

[184] V. R. Vemuri. *Artificial neural networks: Concepts and control applications.* IEEE Computer Society Press, Los Alamitos,CA, 1992.

[185] D. D. Wackerly, W. Mendenhall III, and R. L. Scheaffer. *Mathematical statistics with applications.* Duxbury Press, fifth edition, 1996.

[186] G. M. Weiss and F Provost. The effect of class distribution on classifier learning. Technical Report ML-TR-44, Department of Computer Science, Rutgers University, 2001.

[187] D. Wettschereck. A study of distance-based machine learning algorithms. Ph.D. Thesis, Oregon State University, Corvallis, OR, 1994.

[188] R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. In V. R. Vemuri, editor, *Artificial neural networks: Concepts and control applications*, pages 300–309, Los Alamitos,CA, 1992. IEEE Computer Society Press.

[189] R. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. Rumelhart, editors, *Backpropagation: Theory, Architectures and Applications*, pages 433–486, Hillsdale, NJ, 1995. Lawrence Erlbaum Associates.

[190] L. S. Yaeger, B. J. Webb, and R. F. Lyon. Combining neural networks and context-driven search for online, printed handwriting recognition in the NEWTON. *AI Magazine*, 19(1):73–90, 1998.

[191] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 609–616, San Francisco, CA, 2001. Morgan Kaufmann.

APPENDICES

Appendix A

UCI DATA SETS

This appendix describes the UC,Irvine (UCI) repository data sets as they were used in our experiments. The UCI repository is accessible via the World Wide Web at the URL `http://www.ics.uci.edu/~mlearn/MLRepository.html` or by anonymous ftp from `ftp.ics.uci.edu`. The documentation in the repository describes further details on these data sets.

- Abalone. The task is to predict the age of abalone from physical measurements. In its original form, the domain has 29 classes labeled with 1,2,...29. We have removed the classes that had fewer than six instances: class 1, class 2, class 24, class 25, class 26, class 27, class 28, and class 29.
- Audiology (audio). The original owner of this database is Professor Jergen from the Baylor College of Medicine in Houston. In its original form, the domain has 24 classes (representing different hearing disorders). We have removed the classes that were represented by fewer than five instances.
- Breast Cancer Yugoslavia (bcy). The task is to predict whether a patient has recurrence events in breast cancer. The data was donated by the University Medical Centre, Institute of Oncology, Ljubljana, Slovenia. There are two classes: *Recurrence events* and *No recurrence events.*
- Glass identification (glass). The task involves identifying different types of glass for forensic science investigations. There are six classes.
- Hepatitis. The task is to predict whether hepatits patients will die or not. There are two classes: live and die.
- Iris plant (iris). The task is to classify iris plants. There are three classes: Iris Setosa, Iris Versicolor, and Iris Virginica.
- Liver disorders (liver). The task is to identify patients with liver disorders. There are two classes: sick and healthy.
- Lung cancer (lung). The task is to classify three types of pathological lung cancer.
- Lymphography (lympho). The task is to classify different types of lymphoma cancer. The data was donated by the University Medical Centre, Institute of Oncology, Ljubljana, Slovenia. There are four classes: one is healthy and the other three represent different types of lymphoma.

• Segmentation (segment). The task is to classify objects found in seven outdoor images. The data was donated by Carla Brodley, who was at that time with the Vision Group at the University of Massachussetts, Amherst. There are seven classes: brickface, sky, foliage, cement, window, path, and grass.

• Sonar. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. There are two classes: metal and rock.

• Soybean large (soybean). The task is to identify soybean diseases. The original data has 19 classes. However, virtually all machine learning research studies use only the first 15. The experiments described in this dissertation also employed the 15-class data set.

• Vehicle. The task is to classify vehicles based on their silhouettes. There are four classes: Opel, Saab, Van, Bus.

• Voting records (voting). The task is to find out the party affiliation of the members of the U.S. House of Representatives based on their votes (in 1984). There are two classes: Republican and Democrat.

• Wine. The task is to identify the origin of wines based on chemical analysis. There are three classes.

• Zoology (zoo). The task is to classify different categories of animals. There are seven classes: mammals, birds, reptiles, fish, frogs, insects, marine animals (other than fish).

Appendix B

ALGORITHMS

This appendix describes two algorithms that were cited in this dissertation: Bagging and MetaCost.

Bagging

The first algorithm is Breiman's Bagging [27], an ensemble learning algorithm that relies on voting hypotheses that were obtained by training a learning algorithm on bootstrap replicates of the training set. Table B.1 gives the pseudo-code for the algorithm.

Lines [1] through [4] construct T (given as an input parameter) bootstrap samples of the training data (sampled with replacement from the tarining set) and the classifiers $h_1, h_2, \ldots h_T$ are learned using a classification algorithm. To output the label of a perviously unseen example (line [5]) the algorithm computes the label that is predicted by the majority of the classifiers $h_1, h_2, \ldots h_T$.

TABLE B.1: The BAGGING algorithm. The formula $[\![E]\!] = 1$ if E is true $[\![E]\!] = 0$ otherwise.

Input: a set S, of m labeled examples:
$S =< (x_i, y_i), i = 1, 2, \ldots, m >$,
labels $y_i \in Y = \{1, \ldots, K\}$,
ClassLearn (a classification algorithm)

[1] **for** $t = 1$ **to** T **do**
[2] $S_t :=$ Bootstrap sample of S (sampled with replacement from S);
[3] $h_t := ClassLearn(S_t)$;
[4] **endfor**

Output: $h_f(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{t=1}^{T} [\![h_t(x) = y]\!]$
//the majority label of hypotheses h_t

Bagging can be transformed into a class probability estimation algorithm in two ways:

- 1. If *ClassLearn* can not compute class probability estimates then

$$P(y_i|\mathbf{x}) = \frac{V(y_i)}{T}, 1 \leq i \leq K,$$

where $V(y_i)$ is the number of classifiers that predicted y_i as the label of $\mathbf{x}$ (in other words, the number of votes received by y_i).

- 2. If *ClassLearn* can compute its own class probability estimates $p_t(y_i|\mathbf{x})$, then

$$P(y_i|\mathbf{x}) = \frac{\sum_{t=1}^{T} p_t(y_i|\mathbf{x})}{T}, 1 \leq i \leq K.$$

TABLE B.2: The METACOST algorithm.

Input: a set S, of m labeled examples:
$S =< (x_i, y_i), i = 1, 2, \ldots, m >$,
labels $y_i \in Y = \{1, \ldots, K\}$,
a $K \times K$ cost matrix C,
ClassProbEstim - an algorithm that computes class probability estimates
ClassLearn - a classification algorithm

[1] **for** $i = 1$ **to** m **do**
[2] $\quad \mathbf{P}(y|x_i) = ClassProbEstim(S)$;
[3] $\quad y_i := \underset{y \in Y}{\operatorname{argmin}} \sum_{j=1}^{K} P(j|x_i)C(y, j)$;
[3] **endfor**
Output: $h(x) = ClassLearn(S)$; //the classifier trained on the relabeled data

MetaCost

The second algorithm presented here is Domingos' MetaCost [59] a cost-sensitive learning algorithm that relies on accurate class-probability estimates on the training

data to relabel it and then retrain the classifier to compute the output hypothesis. MetaCost first computes class probability estimates on the training examples. Next it relabels each training example with the optimal class (based on the estimation). The algorithm outputs the hypothesis that is computed by a classification algorithm that is trained on the relabeled data. The pseudo-code for MetaCost is given in Table B.2.

In its original version, MetaCost used modified Bagging (as described in Section B) as *ClassProbEstim*.