UNIVERSITY OF CALIFORNIA, SAN DIEGO

Policy mining:
Learning decision policies from fixed sets of data

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Bianca Zadrozny

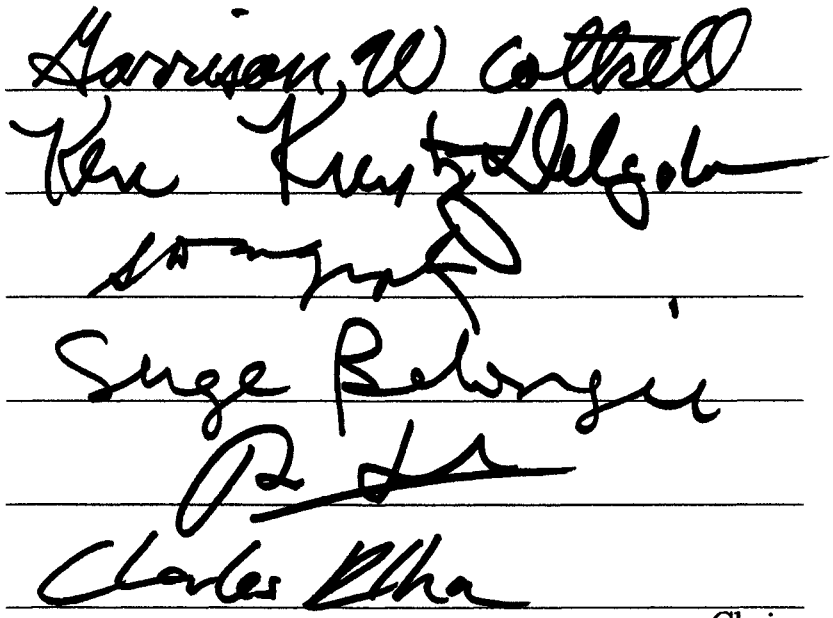
Committee in charge:

Professor Charles P. Elkan, Chair
Professor Serge J. Belongie
Professor Garrison W. Cottrell
Professor Sanjoy Dasgupta
Professor Kenneth Kreutz-Delgado
Professor Padhraic Smyth

2003

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

The dissertation of Bianca Zadrozny is approved, and it is acceptable in quality and form for publication on microfilm:



Chair

University of California, San Diego

2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor Charles Elkan not only for guiding the research presented in this thesis, but also for being a constant source of encouragement throughout my Ph.D. student years.

I give warm thanks to my two other main collaborators: Naoki Abe and John Langford. The work presented in Chapters IV and VII was done in direct collaboration with them and would not have been possible without our many e-mail exchanges.

Thanks also go to the students who were most present in the AI lab at UCSD while I was there: Sameer Agarwal, Kristin Branson, Greg Hamerly, Dana Dahlstrom and Eric Wiewiora. They were great colleagues and provided me with a friendly environment to work in.

I would like to thank my fiancé Roberto Oliveira for helping me believe that I was able to complete this thesis, and for inspiring discussions about the topic of this thesis and many other scientific topics. He also helped me to transform informal mathematical ideas into the theorems presented in Chapters VI and VII.

Finally, I would like to thank my mother Gisela, my father Ivo, my brother André and my grandparents, Diva and Horácio, for being a present, loving and supportive family throughout my life.

The text of Chapter III, in part, is a reprint of the material as it appears in the *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* [95]. The dissertation author was the primary author, and the co-author listed in this publication directed and supervised the research which forms the basis for the chapter.

The text of Chapter IV, in part, is a reprint of the material as it will appear in the *Proceedings of the 2003 IEEE International Conference on Data Mining* [97]. The dissertation author was the primary author, and the co-authors listed in this publication directed and supervised the research which forms the basis for the chapter.

The text of Chapter V, in part, is a reprint of the material as it appears in the *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* [96]. The dissertation author was the primary author, and the co-author listed in this publication directed and supervised the research which forms the basis for the chapter.

I am grateful to the National Science Foundation and IBM Research for financially supporting my Ph.D. studies.

VITA

March 20, 1977	Born, Philadelphia, Pennsylvania
1998	Engineer, Pontificia Universidade Catolica do Rio de Janeiro, Brazil
2001	M.S., University of California, San Diego
2003	Doctor of Philosophy University of California, San Diego

PUBLICATIONS

B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. To appear in *Proceedings of the 2003 IEEE International Conference on Data Mining*, IEEE Computer Society, 2003.

N. Abe, E. Pednault, H. Wang, B. Zadrozny, W. Fan and C. Apte. Empirical comparison of various reinforcement learning strategies for sequential targeted marketing. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 3-10, IEEE Computer Society, 2002.

B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 694-699, ACM Press, 2002.

E. Pednault, N. Abe, B. Zadrozny and others. Sequential cost-sensitive decision-making with reinforcement learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 259-268, ACM Press, 2002.

B. Zadrozny. Reducing multiclass to binary by coupling probability estimates. In *Advances in Neural Information Processing Systems 14*, pages 1041-1048. The MIT Press, 2002.

B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 204-213. ACM Press, 2001.

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*, pp. 609-616. Morgan Kaufmann Publishers, Inc, 2001.

ABSTRACT OF THE DISSERTATION

Policy mining:
Learning decision policies from fixed sets of data

by

Bianca Zadrozny
Doctor of Philosophy in Computer Science
University of California, San Diego, 2003
Professor Charles P. Elkan, Chair

In this thesis we present a new data mining methodology for extracting decision policies from datasets containing descriptions of interactions with an environment. This methodology, which we call policy mining, is valuable for applications in which experimental interaction is not feasible but for which fixed sets of collected data are available. Examples of such applications are direct marketing, credit card fraud detection, recommender systems and medical treatment.

Recent advances in classifier learning and the availability of a great variety of off-the-shelf learners make it very attractive to use classifier learning as the core generalization tool in policy mining. However, in order to successfully apply classifier learning methods to policy mining, three important improvements to the current classifier learning technology are necessary.

First, standard classifier learners assume that all incorrect predictions are equally costly. This thesis presents two general methods for cost-sensitive learning that take into account the fact that misclassification costs are different for different examples and unknown for some examples. The methods we propose are evaluated carefully with experiments using large, difficult and highly cost-sensitive datasets from the direct marketing domain.

Second, most existing learning methods produce classifiers that output ranking scores along with the class label. These scores, however, are classifier

dependent and cannot be easily combined with other sources of information for decision-making. This thesis presents a fast and effective calibration algorithm for transforming ranking scores into accurate class membership probability estimates. Experimental results using datasets from a variety of domains shows that the method produces probability estimates that are comparable to or better than the ones produced by other methods.

Finally, learning algorithms commonly assume that the available data consists of randomly drawn examples from the same underlying distribution of examples about which the learned model is expected to make predictions. In many situations, however, this assumption is violated because we do not have control over the data gathering process. This thesis formalizes the sample selection bias problem in machine learning and presents methods for learning and evaluation under sample selection bias.

Chapter I

Introduction

I.A The policy mining problem

The ability to store and process large amounts of data in computers has increased enormously in the last decade. This has opened up the possibility of analyzing very large datasets gathered from a particular domain of interest to obtain useful knowledge about that domain, in a process known as data mining. Useful knowledge is a very broad concept, and indeed, there are many flavors of data mining that aim to extract different types of knowledge from different types of data.

In some domains, the data can be organized into a set of examples, each represented by a feature vector and a label. One type of useful knowledge that may be extracted in this case is a model that predicts the label of an example when given the values of the features for that example. There are a variety of methods for accomplishing this, collectively known as supervised learning methods [40]. If the label is discrete-valued, the model is known as a classifier.

Classifiers are often used for decision-making, that is, for choosing which action to perform in a particular state described by the feature values. It is straightforward to learn classifiers for this purpose if we have examples of the form $\langle s, a^* \rangle$, where s is a state and a^* is the optimal action for state s. For example, in charac-

ter recognition, s is the image of a character and a^* is the character. The optimal action for a system that receives s as input is to output a^*. Using this type of data we can learn a classifier to predict a^* given s.

However, in many real-world applications, the data consists of a set of examples of the form $\langle s, a, r \rangle$, where s is a state, a is an action executed in the state and r is a reward associated with the action. The reward is a real number that indicates the desirability of action a in state s and it is, in general, stochastic. The optimal action is the one that yields the largest expected reward. Since the optimal action associated with a state is not written explicitly in the data, we cannot learn to predict it directly. Furthermore, the state space is often large (or infinite) so we cannot expect to have data describing each possible action in each possible state.

An example of such an application is a recommender system used by online merchants. In this case, s is the description of a customer (which may include, for example, past purchases), a is a recommendation to buy a product and r is the profit on the product if the customer accepted the recommendation or 0 if he did not accept it. It is clear that the merchant can collect data of the form $\langle s, a, r \rangle$, but not data of the form $\langle s, a^* \rangle$. In the latter case, it would be necessary to recommend each possible product to each customer, which is not feasible. Still, we would like to have a model that predicts the optimal recommendation for a customer in a given state.

Furthermore, in some cases, we may be interested in choosing not just one action but a sequence of actions, such that the total expected reward is maximized. Accordingly, the data consist of examples of the form $(\langle s_0, a_0, r_0 \rangle, \langle s_1, a_1, r_1 \rangle, \ldots, \langle s_n, a_n, r_n \rangle)$. This data indicate that after the execution of action a_i in state s_i, a reward r_i was received and a transition to state s_{i+1} occurred. Because an action influences not only the immediate reward, but all subsequent rewards through the state transition, we have to take this into consideration when choosing the optimal action for a state. Going back to the recommender system example, it

is often the case that a customer interacts multiple times with the system. Each recommendation from the system may influence not only immediate purchases, but future purchases also.

Reinforcement learning is a general framework for learning sequential decisions policies through interaction with an environment [78]. However, current reinforcement learning algorithms are not data mining algorithms because they require direct interaction with the environment or with a simulator of the environment. That is, given an arbitrary state-action pair $\langle s_i, a_i \rangle$, they assume that we can obtain the (stochastic) values for the immediate reward r_i and the next state s_{i+1}.

In this thesis, I will present a new data mining methodology for extracting decision policies from datasets containing descriptions of interactions with an environment. This methodology, which we call *policy mining*, is very valuable for applications in which experimental interaction with the environment is not feasible and no simulator exists, but for which data is available. Examples of such applications are direct marketing, credit card fraud detection, recommender systems and medical treatment.

I.B General approach

Over the last few years there has been a great amount of research effort concentrated in improving classifier learning technology. The community has enjoyed success in improving learning methods considerably, particularly with the advent of support vector machines [72] and ensemble learning methods such as boosting [33] and bagging [14],

We would like to be able to transfer the state-of-the-art in classifier learning to the policy mining problem. Therefore, our general approach to solving the policy mining problem consists in devising appropriate reductions from policy mining to classifier learning. By taking this direction, we can make use of existing (and

future) classifier learning technology as the core generalization component in our methods. In this way, we avoid having to deal directly with difficulties such as preventing overfitting and dealing with very large feature spaces, which have been the main subjects of research in classifier learning. Note that we assume the existence of a classifier learner as a black box, that is, our methods do not depend on the details of any particular classifier learning method.

In classifier learning, we are given a training set of examples of the form (x, y), where x is a feature vector and y is a class label. These examples are assumed (at least, implicitly) to be drawn independently from a fixed distribution D with domain $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is a feature space and $\mathcal{Y}$ is a (discrete) class label space.

Classifier learning methods aim at learning a predictor $h : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the expected error rate on examples drawn from D, given by

$$E_{x,y\sim D}[I(h(x) \neq y)]$$

where $I(\cdot)$ is the indicator function that has value 1 in case its argument is true and 0 otherwise.

The basic idea of reducing policy mining to classifier learning is to use classification for action prediction, where the input for classification is a state $(S = X)$ and the label space is the set of actions $(A = Y)$. This seems simple enough, but we argue that in order to use classifier learning methods in the policy mining setting, we first need to make three important improvements to classifier learning:

1. **Cost-sensitive learning.** Traditional classifier learning algorithms assume that all incorrect predictions are equally costly. However, this assumption is not true in many application areas such as direct marketing and medical treatment. Furthermore, because the distribution of classes is highly skewed in such domains, methods that do not take costs into account fail to identify the less frequent, but more costly, cases. In cost-sensitive learning,

we change this assumption by attempting to minimize example-dependent misclassification costs instead of minimizing error rate.

2. **Calibration of classifier scores.** Most existing learning methods produce classifiers that output scores, which can be used for ranking examples from the most likely member of a class to the least likely member of the class. These scores, however, are classifier dependent and cannot be easily combined with other sources of information, such as domain knowledge, the outputs of other classifiers or misclassification costs. The goal in calibrating classifier scores is to obtain class membership probability estimates that can be used for decision-making.

3. **Sample selection bias correction.** Learning algorithms commonly assume that the available data consists of randomly drawn examples from the same underlying distribution of examples about which the model is expected to make predictions. In many applications, however, this assumption is violated because we do not have control over the data gathering process. Sample selection bias correction methods aim at learning a predictor from a biased sample that is as accurate as possible for the true underlying distribution. Furthermore, they should allow us to estimate the accuracy for the underlying distribution using the available data.

The exact role that these subproblems play in solving the general problem of extracting decision policies from data will need to be delayed until Chapter VII where we define the policy mining setting more formally and present an algorithm for policy mining. Note, however, that these are important improvements to classifier learning in themselves and can serve as building blocks for solving other learning problems.

I.C Overview of the dissertation

In Chapter II we give an introduction to cost-sensitive learning. We present the two existing formulations for introducing misclassification costs into a classifier learning problem, namely the *cost matrix* formulation and the *importance formulation.* We explain the connections between the two and their respective advantages and disadvantages. We also give a brief overview of current research in cost-sensitive learning and give a detailed description of two public available cost-sensitive learning datasets that are used for experiments in the subsequent chapters on cost-sensitive learning methods (Chapters III and IV).

In Chapter III we present *direct cost-sensitive decision making,* a general method for cost-sensitive learning that uses the cost matrix formulation and is based on expected cost estimation. We compare it to MetaCost, due to Domingos [20], which was the first method for transforming any classifier learning method into a cost-sensitive learner. Because we allow for example-dependent costs and unknown costs for some examples, direct cost-sensitive decision making is more general than MetaCost as originally published. Furthermore, our experimental results show that it is preferable to MetaCost.

In Chapter IV we present a family of cost-sensitive learning methods that uses the importance formulation. These methods are based on a reduction from cost-sensitive learning to classifier learning that requires a change in the distribution of training examples. In particular, we propose *costing,* an ensemble learning method that uses rejection sampling to produce a cost-sensitive classifier using only black box access to a classifier learning method. Costing does not require accurate class membership probability estimates from the classifier and avoids the estimation of costs, so it is conceptually simpler than the methods for cost-sensitive learning by expected cost estimation. Nonetheless, our experimental results show that its performance is comparable to direct cost-sensitive decision making for a variety of classifier learning methods.

In Chapter V, we motivate the need for calibrated class membership probability estimates and present a new method for obtaining calibrated two-class probability estimates that can be applied to any classifier that produces a ranking of examples. Besides being fast and very simple to understand and implement, our method produces probability estimates that are comparable to or better than the ones produced by other methods, such as the one proposed by Platt [63]. We also present the first method for obtaining calibrated probability estimates from ranking scores for multiclass problems. We demonstrate that by decomposing the multiclass problem into two-class problems, obtaining calibrated probability estimates for each problem and correctly combining these probability estimates we can obtain calibrated multiclass probability estimates.

In Chapter VI we formally define the sample selection bias problem in machine learning terms. We then study the behavior of a number of well-known classifier learning methods under sample selection bias. For classifier learning methods that are affected by sample selection bias, we present a correction method based on estimating the probability that an example is selected into the sample and using rejection sampling to modify the distribution of examples. Finally, we consider the problem of evaluating a classifier using a selected sample and present a method for obtaining an unbiased estimate of the performance of a classifier using a biased sample of test examples.

In Chapter VII we give an overview of reinforcement learning and formally define the policy mining problem using the Markov Decision Process (MDP) framework that is commonly used in reinforcement learning. We argue that the current reinforcement learning methodology is not suitable for solving the policy mining problem and present a new formulation that we call *reinforcement learning with traces*. This formulation does not require the availability of a simulator for the environment and, instead, uses a trace model that can be simulated with fixed sets of data collected offline. We show that for one-step MDPs, we can reduce reinforcement learning with traces to cost-sensitive learning with sample selection

bias correction. For MDPs with arbitrary number of steps, we present a greedy iterative method that learns a cost-sensitive classifier for each step. The policy obtained with this method is the approximately best possible local improvement over the arbitrary policy used for collecting the data.

Chapter II

Cost-sensitive learning

In this chapter we define the cost-sensitive learning problem by presenting two alternative ways of introducing misclassification costs into the standard classifier learning problem and showing the connections between them. We then give a brief survey of current methods for cost-sensitive learning. Finally, we give an overview of the publicly available datasets collected from real-world cost-sensitive domains that can be used for running cost-sensitive learning experiments.

II.A Introducing costs into classifier learning

In standard classifier learning, we are given a training set of examples of the form (x, y), where x is a feature vector and y is a class label. These examples are assumed (at least, implicitly) to be drawn independently from a fixed distribution D with domain $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is a feature space and $\mathcal{Y}$ is a (discrete) class label space. The goal is to learn a classifier $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the expected error rate on examples drawn from D, given by

$$E_{x,y\sim D}[I(h(x) \neq y)] \tag{II.A.1}$$

where $I(\cdot)$ is the indicator function that has value 1 in case its argument is true and 0 otherwise.

The traditional formulation assumes that all errors are equally costly.

However, this assumption is not true for many domains for which one would like to obtain classifiers. For example:

- In one-to-one marketing, the cost of making an offer to a person who does not respond is typically small compared to the cost of not contacting a person who would respond.
- In medicine, the cost of prescribing a drug to an allergic patient can be much higher than the cost of not prescribing the drug to a nonallergic patient, if alternative treatments are available.
- In image or text retrieval, the cost of not displaying a relevant item may be lower or higher than the cost of displaying an irrelevant item.
- For most animals, failing to recognize a predator and hence not fleeing is far more costly than fleeing from a non-predator.

In the following sections we will present two alternative approaches for introducing costs into the classifier learning problem.

II.A.1 The cost matrix formulation

One extension to the standard classifier learning formulation that has received considerable attention in the past few years is the *cost matrix* formulation [28, 20, 58].

In this formulation, besides assuming the availability of a training set, we specify a cost matrix C for the domain in which we would like to learn a classifier. If there are k classes, the cost matrix is a $k \times k$ matrix of real values. Each entry $C(i, j)$ gives the cost of predicting class i for an example whose actual class is j. Although for many applications the diagonal entries $C(i, i)$ are zero, this does not necessarily have to be the case, since predicting one class correctly may be more important than predicting another class correctly.

Table II.1 shows a cost matrix for a 3-class problem. These costs were obtained by assuming that a person uses the classifier before leaving to work to

predicted class	actual class		
	sunny	snowy	rainy
sunny	0	10	15
snowy	1	1	11
rainy	2	2	2

Table II.1: Example of cost matrix for a weather classification problem.

predict if the weather will be sunny, rainy or snowy that day, and the following is true:

1. The person takes the jacket when the prediction is either rainy or snowy.
2. The person takes the umbrella only when the prediction is rainy.
3. Taking the jacket incurs a cost of 1.
4. Taking the umbrella incurs a cost of 1.
5. Not having a jacket when it is snowy incurs a cost of 10.
6. Not having a jacket when it is rainy incurs a cost of 5.
7. Not having an umbrella when it is rainy incurs a cost of 10.

In this cost matrix, two of the diagonal entries are not zero. This is the case because there are fixed costs associated with predicting snowy and rainy weather, but not for sunny weather.

Now, instead of minimizing the error rate given by equation II.A.1, we would like to find a classifier h that minimizes the expected cost of the labeling, given by

$$E_{x,y\sim D}[C(h(x),y)]. \tag{II.A.2}$$

Note that if C is the identity matrix this reduces to equation II.A.1.

Not all cost matrices are logically reasonable. Elkan [28] gives reasonableness conditions for cost matrices and explains how to avoid specifying contradictory cost matrices by measuring all costs against a fixed baseline.

Research on cost-sensitive learning has traditionally been couched in terms of costs, as opposed to benefits or rewards. However, in many domains, it is easier to talk consistently about benefits than about costs. The reason is that all benefits are straightforward cash flows relative to a baseline wealth of \$0, while some costs are counterfactual opportunity costs [28]. For these domains, we can specify a benefit matrix B, where each entry of the matrix describes the benefit (or reward) of predicting class i for an example whose actual class is j. Then, instead of minimizing II.A.2, we maximize

$$E_{x,y\sim D}[B(h(x), y)].$$

Fixed vs. example-dependent costs

The standard cost matrix formulation assumes that the misclassification costs are fixed, i.e., that they only depend on the predicted and actual classes, but not on the example itself. However, more often than not, misclassification costs in real-world domains are example-dependent. For example, in direct marketing, the cost of mislabeling a respondent as a non-respondent depends on the profit that the customer would have generated. Similarly, in credit card fraud detection, the cost of mislabeling a fraud transaction as a non-fraud transaction depends on the amount of the transaction.

Previous research in cost-sensitive learning has primarily been focused on the case of fixed costs described by a cost matrix. In this dissertation, however, we present learning methods for the more general example-dependent case. We can formalize this case by extending the cost matrix formulation.

We extend the cost matrix formulation to the example-dependent case by allowing each entry to depend on the particular feature vector x. In this case, the costs are given by a function $C(i, j, x)$, where i is the predicted class, j is the actual class and x is the feature vector of the example. Accordingly, we would now

predicted class	actual class non-fraud	fraud
non-fraud	0	$t(x)$
fraud	a	a

Table II.2: Example of cost matrix for a credit card fraud detection problem, where a is the (fixed) cost of auditing a credit card transaction and $t(x)$ is the transaction amount for example x.

like to find a classifier h that minimizes the expected cost of the labeling, given by

$$E_{x,y\sim D}[C(h(x), y, x)]. \tag{II.A.3}$$

Table II.2 shows an example-dependent cost matrix for a 2-class credit card fraud detection problem. In this domain, the costs are assigned as follows. If we predict fraud, the transaction will be audited, which incurs a fixed cost a. On the other hand, if we predict non-fraud, the transaction will not be audited. In this case, the cost is zero if the transaction is not a fraud, and it is the transaction amount $t(x)$ if the transaction is a fraud.

Known vs. unknown costs

Because the standard cost matrix formulation assumes that the misclassification costs are the same for all examples, it also implicitly assumes that they are known in advance for all examples. However, when we allow example-dependent costs, it might be the case that the costs are not known for some of the examples. In particular, it is very common for costs to be known for the examples in the training set but not for new unlabeled examples to which we would like to apply the classifier (test examples).

Interestingly, similar problems in terms of the structure of the cost matrix may have different properties in terms of whether the costs are known or unknown. Table II.3 shows an example-dependent cost matrix for a 2-class donation request problem. The costs are assigned as follows. If we predict non-donor, we will not

predicted class	actual class non-donor	donor
non-donor	0	0
donor	m	$m - d(x)$

Table II.3: Example of cost matrix for a donation request problem, where m is the (fixed) cost of mailing a request and $d(x)$ is the donation amount for example x.

mail a donation request, so the cost is always zero. If we predict donor we will mail a donation request, so there is always a fixed cost of m for mailing and, in the case of a donation, there is a negative cost (or a benefit) corresponding to the donation amount $d(x)$.

This cost matrix is similar to the one in table II.2 in the sense that there is only one entry for which the cost is example-dependent. However, in the credit card fraud detection problem the cost is known for all examples (including unlabeled examples) because the transaction amount $t(x)$ is available at the time we apply the classifier to decide whether the transaction is fraudulent or not. On the other hand, in the donation request problem, the value of the donation amount $d(x)$ is unknown for all examples in the test set, and only known for examples in the training set that correspond to donors.

II.A.2 The importance formulation

For the two-class case, there is an alternative formulation for cost-sensitive learning that we call the importance formulation. Here we assume that each training example is associated with a real number, the *importance* of the example, corresponding to the difference in cost between classifying the example incorrectly and correctly. Because we assume that it is always at least as costly to classify the example incorrectly than correctly, importances are always nonnegative numbers (see reasonableness conditions [28]).

More formally, we assume that the examples are drawn independently from a distribution D with domain $\mathcal{X} \times \mathcal{Y} \times \mathcal{C}$ where X and Y are the same as

in the standard classifier learning formulation and $\mathcal{C} \subset [0, \infty]$ is the importance (or extra cost) associated with mislabeling that example. The goal is to learn a classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$ which minimizes the expected cost of the labeling

$$E_{x,y,c\sim D}[c\, I(h(x) \neq y)] \tag{II.A.4}$$

given training data of the form (x, y, c).

This formulation naturally models example-dependent and noisy costs. It is also simpler and easier to manipulate mathematically than the cost matrix formulation. Because the cost is part of the example, we do not have to deal with a separate mathematical entity. The drawback of the importance formulation is that it is only suitable for two-class problems. Formulating a cost-sensitive learning problem in a similar way when there are more than two classes is an open problem.

For application domains in which it is easier to talk consistently about benefits (or profits) than about costs, we can have the importance be the difference in benefit between classifying the example correctly and incorrectly. Again, we assume that classifying the example correctly has a larger benefit than classifying it incorrectly, so that the importance is positive. Then, maximizing equation II.A.4 will correspond to maximizing the expected profit.

We note that there is a recent trend in cost-sensitive learning research moving from the cost matrix formulation to the importance formulation [36, 13, 97].

II.A.3 Relationship between the two formulations

As it turns out, we can represent any two-class (example-dependent) cost matrix using the importance formulation. This is true because given the cost matrix and an example, only two entries (false positive, true negative) or (false negative, true positive) are relevant for that particular example, depending on whether the example is negative or positive. In fact, if we let the importance be

$$c = C(1, 0, x) - C(0, 0, x)$$

for negative examples and

$$c = C(0,1,x) - C(1,1,x)$$

for positive examples (0 stands for negative and 1 stands for positive), we can show that minimizing the expected cost in the cost matrix formulation is equivalent to minimizing the expected cost in the importance formulation.

Theorem II.A.1. *Let C be a two-class cost matrix and let (x,y) be examples independently drawn from a distribution D with domain $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is a feature space and $\mathcal{Y}$ is a (binary) label space. For each example (x,y), let the corresponding importance be $c = C(1,0,x) - C(0,0,x)$ if $y = 0$ and $c = C(0,1,x) - C(1,1,x)$ if $y = 1$. Then*

$$min_h E_{x,y\sim D}[C(h(x),y,x)] = min_h E_{x,y\sim D}[c\, I(h(x) \neq y)].$$

Proof.

$$\begin{aligned}
&E_{x,y\sim D}[C(h(x),y,x)] \\
&= E_{x,y\sim D}[C(h(x),0,x)I(y=0) + C(h(x),1,x)I(y=1)] \\
&= E_{x,y\sim D}[(C(0,0,x)I(h(x)=0) + C(1,0,x)I(h(x)=1))I(y=0) \\
&\qquad + (C(0,1,x)I(h(x)=0) + C(1,1,x)I(h(x)=1))I(y=1)] \\
&= E_{x,y\sim D}[(C(0,0,x)(1 - I(h(x)\neq 0) + C(1,0,x)I(h(x)\neq 0))I(y=0) \\
&\qquad + (C(0,1,x)I(h(x)\neq 1) + C(1,1,x)(1 - I(h(x)\neq 1))I(y=1)] \\
&= E_{x,y\sim D}[(C(1,0,x) - C(0,0,x))I(h(x)\neq 0)I(y=0) \\
&\qquad + (C(0,1,x) - C(1,1,x))I(h(x)\neq 1)I(y=1) + C(0,0,x) + C(1,1,x)] \\
&= E_{x,y\sim D}[c\, I(h(x)\neq 0)I(y=0) + c\, I(h(x)\neq 1)I(y=1) \\
&\qquad + C(0,0,x) + C(1,1,x)] \\
&= E_{x,y\sim D}[c\, I(h(x)\neq y) + C(0,0,x) + C(1,1,x)]
\end{aligned}$$

Since the term $C(0,0,x) + C(1,1,x)$ does not influence the minimization with respect to h, we have that

$$\begin{aligned}
&\min_h E_{x,y\sim D}[C(h(x),y,x)] \\
&= \min_h E_{x,y\sim D}[c\, I(h(x)\neq y) + C(0,0,x) + C(1,1,x)] \\
&= \min_h E_{x,y\sim D}[c\, I(h(x)\neq y)]
\end{aligned}$$

which completes the proof. □

Although they are equivalent, the cost matrix formulation and the importance formulation lend themselves naturally to two different general approaches for solving the cost-sensitive learning problem, respectively:

- Cost-sensitive learning by expected cost estimation
- Cost-sensitive learning by example weighting

We will explore each of these approaches in Chapters III and IV.

II.B Current approaches to cost-sensitive learning

Research in cost-sensitive learning falls into three categories. The first category is concerned with making particular classifier learners cost-sensitive. Below are examples of work in this category for a variety of learning methods:

- Decision trees: Knoll et al. [48] and Bradford et al. [12] present cost-sensitive pruning methods for decision trees, while Drummond and Holte [22] investigate the effect of splitting criteria on cost-sensitive learning of decision trees.
- Boosting: Fan et al. [31] propose AdaCost, a misclassification cost-sensitive boosting method.
- Neural networks: Geibel and Wysotski [36] propose a cost-sensitive perceptron learning rule for non-separable classes.
- Support vector machines: Fumera and Roli [35] and Brefeld et al. [13] propose cost-sensitive support vector machine learning algorithms.

The second category uses Bayes risk theory to assign each example to its lowest expected cost class [20, 95, 58]. This requires classifiers to output class membership probabilities and, in the case where costs are example-dependent and unknown for some examples, also requires estimating costs [95]. These methods,

which we call *cost-sensitive learning by expected cost estimation*, will be covered in detail in Chapter III.

The third category concerns methods that modify the distribution of training examples before applying the classifier learning method, so that the classifier learned from the modified distribution is cost-sensitive. This class of methods, which we call *cost-sensitive learning by example weighting* has only been explored for fixed costs in previous research [57]. In Chapter IV we present and evaluate such a method for example-dependent costs.

Besides misclassification costs, there may be other types of costs involved in classifier learning that we do not consider in this thesis, such as the cost of measuring attributes and the cost of labeling new examples. Turney [84] has created a taxonomy of the different types of costs involved in machine learning. Recently, Bayer-Zubek [7] has proposed a method for cost-sensitive learning that takes into consideration both measurement costs and misclassification costs. Note, however, that this is not a method for transforming existing classifier learners into cost-sensitive learners like the ones presented in this thesis, but a learning method specifically designed for this purpose.

II.C Publicly available cost-sensitive datasets

Here we give an overview of the two cost-sensitive datasets that are used for experimentation in this dissertation. Unfortunately, these are the only two publicly available real-world datasets for which misclassification cost information is available on a per example basis. Both datasets are from the direct marketing domain. Although there are many other cost-sensitive classifier learning domains, such as credit card fraud detection and medical treatment, publicly available cost-sensitive datasets are lacking.

Much of the research in cost-sensitive learning has been done using synthetic costs (see for example [20, 31, 36, 13]), which is not very satisfactory given

that real-world costs can exhibit certain peculiar characteristics that may not be captured by synthetic costs. Another option has been the use of proprietary datasets, such as a credit card fraud detection dataset [31, 32], which is also not satisfactory because it impedes comparison of methods developed by different researchers.

We believe that the creation of a dataset repository such as the UCI Machine Learning Archive [10] Archive and the UCI KDD Archive [6] for cost-sensitive datasets would be very valuable for the advancement of cost-sensitive learning methodology.

II.C.1 The KDD-98 dataset

This is a well-studied, large and challenging dataset that was first used in the data mining contest associated with the 1998 KDD conference and is now becoming popular as a benchmark for the evaluation of cost-sensitive learning methods [95, 97, 32]. This dataset and associated documentation are now available in the UCI KDD repository [6].

The dataset contains information about persons who have made donations in the past to a particular charity. The decision-making task is to choose which donors to mail a request for a new donation. in order to maximize the total profit obtained in the mailing campaign. This task is completely analogous to typical one-to-one marketing tasks for many other organizations, both non-profit and for-profit. Mathematically, the task has the same structure as all the two-class cost-sensitive learning problems mentioned in the section II.A.

The KDD-98 dataset is divided in a fixed, standard way into a training set and a test set. The training set consists of 95412 records for which it is known whether or not the person made a donation (a 0/1 response) and how much the person donated, if a donation was made. The test set consists of 96367 records from the same donation campaign for which similar donation information was not published until after the KDD-98 competition. In order to make our experimen-

tal results directly comparable with those of previous work, we use the standard training set/test set division.

Each example in the dataset consists of 481 attributes describing each individual's donation history in the previous 22 campaigns, as well as demographic information. Since this dissertation does not address the issue of feature selection, our choice of attributes is fixed and based informally on the KDD-99 winning submission of Georges and Milley [37]:

- `income`: household income code (range 1–8)
- `firstdate`: date of first gift
- `lastdate`: date of most recent gift
- `pgift`: number of gifts/number of promotions received
- `RFA_2F`: frequency code (range 1–4)
- `RFA_2A`: amount of last gift code (range A–G)
- `PEPSTRFL`: RFA (recency, frequency, amount) star status (X or blank).
- `avggift`: average dollar amount of gifts to date.
- `lastgift`: dollar amount of most recent gift.
- `ampergift`: average dollar amount in responses to the last 22 promotions.

The attributes `pgift` and `ampergift` are not directly present in the KDD98 data, but are obtained by dividing `ngiftall/numprom` and by averaging `RAMNT_3` to `RAMNT_24`, respectively.

Mailing a solicitation to an individual costs the charity \$0.68. The overall percentage of donors among potential recipients is about 5%. The donation amount for persons who respond varies from \$1 to \$200. Given the low response rate and the variation in the value of gifts, it is not easy to achieve a profit that is much higher than that obtained by soliciting all potential donors. The profit obtained

by soliciting every individual in the test set is \$10560, while the profit attained by the winner of the KDD-98 competition was \$14712.

Many participants in the KDD-98 competition submitted entries that were worse than useless, because they achieved profits substantially lower than \$10560. One likely reason for low success is that the individuals in the KDD-98 dataset are already filtered to be a reasonable set of prospects. They have been the targets of a real donation campaign, selected using standard techniques in direct marketing such as recency-frequency-amount (RFA) scoring. The task now for any cost-sensitive learning method is to improve upon the already good performance of the unknown method that was applied to create the KDD-98 dataset.

We now formulate the problem as a cost-sensitive learning problem using both the cost matrix formulation and the importance formulation. Since the goal in this domain is to maximize the profit, we use benefits instead of costs in both formulations.

The benefit matrix formulation

If we predict that the example is a non-donor, we will not mail a solicitation. Thus, in this case, the benefit is fixed at zero. If we predict the example is a donor, we will mail a solicitation, which costs \$0.68. If the person is a donor, we will also receive a benefit corresponding to the donation amount $y(x)$, which is example-dependent. So we have the following benefit matrix B:

	actual non-donor	actual donor
predict non-donor	0	0
predict donor	-0.68	$y(x) - 0.68$

The importance formulation

The importance of each example is the difference in benefit between predicting the class label correctly or predicting it incorrectly.

For negative examples (actual non-donors), the correct prediction will lead to zero benefit while the incorrect prediction will lead to a negative benefit (a

cost) of \$0.68 for mailing. Thus, the importance is fixed at 0-(-0.68)= 0.68 for all negative examples.

For positive examples (actual donors), the correct prediction will lead to a benefit corresponding to the donation amount $y(x)$ minus \$0.68 for mailing, while the incorrect prediction will lead to zero benefit. Thus, for positive examples (actual donors), the importance is $y(x) - 0.68$, which varies from \$0.32 to \$199.32 for the positive examples in the training set.

II.C.2 The DMEF-2 dataset

This dataset can be obtained for research and educational purposes from the DMEF dataset library [2] for a nominal fee. Although it is not as well-known as the KDD dataset, it has been used in previous research in data mining [55].

The dataset contains customer buying history for 96551 customers of a nationally known catalog. The decision-making task is to choose which customers should receive a new catalog so as to maximize the total profit on the catalog mailing campaign. Information on the cost of mailing a catalog is not available, so we fixed it at \$2.

The overall percentage of respondents is about 2.5%. The purchase amount for customers who respond varies from \$3 to \$6247. We divided the dataset randomly in half to create a training set and a test set. As a baseline for comparison, the revenue obtained by mailing a catalog to every individual on the training set is \$26474 and on the test set is \$27584.

Each example consists of more than 150 attributes detailing the history of past catalog purchases of the customer, along with a label indicating whether or not the customer has responded to the last campaign and, in the case of response, the purchase amount in dollars. Again, because we are not considering the problem of feature selection here, we selected the following 17 features using domain knowledge:

- `totalorders`: number of orders

- `totaldollars`: total amount
- `totalorders6`: number of orders in the past 6 months
- `totaldollars6`: total dollars in the past 6 months
- `totalorders12`: number of orders in the past 12 months
- `totaldollars12`: total dollars in the past 12 months
- `totalorders24`: number of orders in the past 24 months
- `totaldollars24`: total dollars in the past 24 months
- `totalorders36`: number of orders in the past 36 months
- `totaldollars36`: total dollars in the past 36 months
- `dayslast`: days since last purchase
- `daysfirst`: days since first purchase
- `rfm`: recency-frequency-monetary score
- `recencylastD`: recency of last purchase in division D
- `recencyfirstD`: recency of first purchase in division D
- `totalitemsD`: total items in division D
- `rfmD`: recency-frequency-monetary score in division D

We use the features for division D because this is the division of the catalog mailing that we are trying to optimize.

The formulation of this problem as a cost-sensitive learning problem is completely analogous to the formulation for the KDD-98 dataset. The benefit matrix is as follows (where $y(x)$ is the purchase amount):

	actual non-buyer	actual buyer
predict non-buyer	0	0
predict buyer	-2	$y(x) - 2$

For positive examples (buyers), the importance is $y(x) - 2$, which varies from \$1 to \$6246 for the examples in the training set. For all negative examples (non-buyers), the importance is fixed at \$2.

Chapter III

Cost-sensitive learning by expected cost estimation

In this chapter, we present and compare two methods for cost-sensitive learning that use the cost matrix formulation and are based on expected cost estimation. One method is MetaCost, due to Domingos [20], which was the first method for transforming any classifier learning method into a cost-sensitive learner. The other is direct cost-sensitive decision making, proposed by Zadrozny and Elkan [94] as an improvement to MetaCost. Our analysis shows that direct cost-sensitive decision making is more general than MetaCost as originally published, and our experimental results show that it is preferable to MetaCost.

This chapter is organized as follows. In Section III.A we explain MetaCost and direct cost-sensitive decision-making. Then in Section II.C.1 we show how to apply these methods to the KDD-98 dataset. Both MetaCost and direct cost-sensitive decision-making require accurate estimates of class membership probabilities. In Section III.C we present two techniques that allow accurate probability estimates to be obtained from a decision tree: smoothing and curtailment. We also present binning as a technique for making naive Bayes probability estimates accurate. Previous research has been based on the assumption that misclassification costs are the same for all examples and known in advance, but in

general these costs are example-dependent and unknown, in the same way that class membership probabilities are example-specific and not known in advance. In Section III.D we discuss this issue and the issue of how sample selection bias affects cost estimation. Finally, experimental results using the KDD-98 dataset are presented in Section III.E and in Section III.F we summarize the main contributions in this chapter. Related work is discussed as necessary throughout the chapter.

III.A MetaCost vs. direct cost-sensitive decision-making

In the cost matrix formulation each training or test example x is associated with a cost $C(i, j, x)$ of predicting class i for x when the true class of x is j. If these costs are known for each x, i and j, and the class membership probabilities $P(j|x)$ are known for each x and j then it is straightforward to compute an optimal policy for decision-making .

The optimal prediction for x, i.e. the optimal decision concerning x or label to assign to x, is the class i that leads to the lowest expected cost

$$\sum_j P(j|x)C(i, j, x). \tag{III.A.1}$$

Given x, for each alternative i, the expected cost is a weighted average where the weight of $C(i, j, x)$ is the conditional probability of the class j given x. The label that leads to the lowest expected cost is known in the literature as the *Bayes optimal prediction* for x [23].

The central idea behind the MetaCost method is to change the label of each training example to be its optimal label according to Equation (III.A.1), and then to learn a classifier that predicts these new labels.

Applying MetaCost requires knowledge of the conditional probability $P(j|x)$ for each training example x and each possible true class j for x. Almost always, these probabilities are not given as part of the training data. Instead, the training data must be used to learn a classifier that estimates $P(j|x)$ for each

MetaCost(Learner A, Training Set S)

1. **Learn a model for $P(j|x)$ using learner A applied to S.**
2. **Relabel each example x in S with**

$$i^* = \text{argmin}_i \sum_j P(j|x)C(i,j,x)$$

   to form the set S'.

3. **Learn a classifier C using learner A applied to S'.**
4. **Output C.**

Table III.1: The MetaCost algorithm.

training example x and each j. Table III.1 gives the pseudo-code for the MetaCost algorithm.

However, note that any learned classifier that can provide conditional class membership probability estimates for training examples can also provide these estimates for test examples. Using these probability estimates we can directly compute the optimal label for each test example using Equation (III.A.1). This process is the method that we call direct cost-sensitive decision-making. The pseudo-code for this method is given in table III.2. Experimental results comparing MetaCost and direct cost-sensitive decision-making are given in Section III.E.

The basic MetaCost idea can be implemented in many ways. Our implementation differs from that described by Domingos [20] in two important ways. First, the original description of MetaCost is based on the assumption that costs are known in advance and are the same for all examples, i.e. that $C(i,j,x) = C(i,j)$ with no dependence on x. Provost and Fawcett [65] have pointed out that this assumption is not always true: "For some problems, different errors of the same type have different costs." We generalize MetaCost by relaxing this assumption.

Second, we do not estimate probabilities by using bagging [14]. Instead of

Direct cost-sensitive decision-making(Learner A, Training Set S)

1. **Learn a model for $P(j|x)$ using learner A applied to S.**
2. **Let C be the classifier that outputs**

$$i^* = \text{argmin}_i \sum_j P(j|x)C(i,j,x)$$

   for an example x.

3. **Output C.**

Table III.2: The direct cost-sensitive decision-making algorithm.

bagging, we use simpler methods based on single decision trees and naive Bayesian classifiers. As pointed out by Margineantu [56], bagging gives voting estimates that measure the stability of the base classifier learning method at an example, not the actual class conditional probability of the example. (A classifier learning method is stable at an example if classifiers learned from different resamples predict the same label for the example). For experimental results confirming that bagging is not a good way of improving probability estimates obtained from decision trees, see Zadrozny and Elkan [95].

In general, bagging does not give probability estimates that are unbiased and well-calibrated, whether or not the base learning method is stable. If a learning method is unstable and gives classifiers that make 0/1 predictions, then bagging tends to be useful because voting estimates are numbers between 0 and 1, which are preferable to 0/1 predictions as continuous probability estimates. However, in general these scores are not unbiased estimates. If a learning method gives classifiers that individually yield unbiased probability estimates, then bagging these classifiers is likely to reduce variance beneficially, while maintaining unbiasedness. But then the question remains of how to get individual scores that are unbiased in the first place. Section 4 below answers this question.

III.B Applying direct cost-sensitive decision making

The dataset used in the experimental work described in this chapter is a well-studied, large and challenging dataset that was first used in the data mining contest associated with the 1998 KDD conference that was described in detail in chapter II (section II.C.1).

Let the label $j = 0$ mean the person x does not donate, and let $j = 1$ mean the person does donate. If the person donates, the donation is of a variable amount, say $y(x)$. The cost of mailing a solicitation is \$0.68, so we have the following benefit matrix $B(i, j, x)$:

	actual non-donor	actual donor
predict non-donor	0	0
predict donor (mail)	-0.68	$y(x) - 0.68$

The optimal predicted label for example x is the class i that maximizes

$$\sum_j P(j|x)B(i, j, x) \tag{III.B.1}$$

where $B(i, j, x)$ is the benefit of predicting class i when the true class is j.

Notice that $B(1, 1, x)$ is example-dependent and unknown for test examples. We shall argue later that no fixed matrix of costs or benefits can lead to good decision-making. There is no constant c such that it would be reasonable to replace $B(1, 1, x)$ by the same value c for all x. All approaches to this task, and to other tasks with the same structure, that are based on a fixed cost or benefit matrix will have poor performance. Of course, some approaches can take into account the fact that $y(x)$ is example-dependent without estimating $y(x)$ explicitly.

The expected benefit of not soliciting a person x, i.e. of deciding $i = 0$ for x, is

$$P(j = 0|x)B(0, 0, x) + P(j = 1|x)B(0, 1, x)$$

$$= P(j = 0|x)(0) + P(j = 1|x)(0)$$

$$= 0$$

The expected benefit of soliciting x is

$$P(j=0|x)B(1,0,x) + P(j=1|x)B(1,1,x)$$

$$= P(j=0|x)(-0.68) + P(j=1|x)(y(x) - 0.68)$$

$$= (1 - P(j=1|x))(-0.68) + P(j=1|x)(y(x) - 0.68)$$

$$= P(j=1|x)y(x) - 0.68.$$

The optimal policy is to solicit exactly those people for whom the expected benefit of mailing is greater than the expected benefit of not mailing: individuals for whom

$$P(j=1|x)y(x) - 0.68 > 0.$$

In other words, the optimal policy is to mail to people for whom the expected return $P(j=1|x)y(x)$ is greater than the cost of mailing a solicitation:

$$P(j=1|x)y(x) > 0.68. \qquad \text{(III.B.2)}$$

In order to apply this policy, we need to estimate the conditional probability of making a donation $P(j=1|x)$ and the donation amount $y(x)$ for each example x in the training set, in the case of MetaCost. We need to estimate these values for both training and test examples in the case of direct cost-sensitive decision-making.

Although we use the KDD-98 dataset for concreteness, the methods described here apply to cost-sensitive learning in general. In any cost-sensitive learning application, in order to use Equation (III.A.1) or (III.B.1) to obtain an optimal labeling, we need to estimate conditional class membership probabilities accurately. Costs or benefits must also be estimated whenever they are unknown for some examples.

In general, if x is a test example then $C(i,j,x)$ will be unknown for all i and j. If x is a training example then $C(i,j,x)$ will be known for some i and j pairs, but unknown for other pairs. Of course, if costs are not example-dependent, that is, if $C(i,j,x) = C(i,j,y)$ for all examples x and y, then costs do not need to

be estimated for any training or test examples. This special case is the only case considered in previous general research on cost-sensitive learning. In the remainder of this chapter, we present new methods for estimating costs and probabilities. All these methods can be applied without change in a wide variety of domains.

III.C Probability Estimation Methods

An estimate of the conditional probability of membership in each class is required for each training example if MetaCost is used, and for each test example if direct cost-sensitive decision-making is used.

This section explains our methods for obtaining calibrated probability estimates from decision tree and naive Bayesian classifiers. We first explain the deficiencies that cause standard decision tree methods not to give accurate probability estimates, and we then explain methods to overcome these limitations. A final subsection presents a simple method for obtaining calibrated probabilities from a naive Bayesian classifier.

III.C.1 Deficiencies of decision tree methods

Throughout this chapter, C4.5 [66] is the representative decision tree learning method used, but all our analyses and suggestions apply equally to other decision tree methods such as CART [15].

When classifying a test example, C4.5 and other decision tree methods assign by default the raw training frequency $p = k/n$ as the score of any example that is assigned to a leaf that contains k positive training examples and n total training examples. These training frequencies are not accurate conditional probability estimates for at least two reasons:

1. High bias: Decision tree growing methods try to make leaves homogeneous, so observed frequencies are systematically shifted towards zero and one. This problem has been noted by Walker [88] and others.

2. High variance: When the number of training examples associated with a leaf is small, observed frequencies are not statistically reliable.

Pruning methods as surveyed by Esposito *et al.* [30] can in principle alleviate problem (2) by removing leaves that contain too few examples. However, standard pruning methods are not suitable for unbalanced datasets, because they are based on accuracy maximization. On the KDD-98 dataset C4.5 produces a pruned tree that is a single leaf. Since the base rate of positive examples, that is the overall probability $P(j = 1)$, is about 5%, this tree has accuracy 95%, but it is useless for estimating example-specific conditional probabilities $P(j = 1|x)$.

In general, trees pruned with the objective of maximizing accuracy are not useful for ranking test examples, or for estimating class membership probabilities. The standard C4.5 pruning method is not alone in being incompatible with accurate probability estimation. Quinlan's recent decision tree learning method, C5.0, and CART also do pruning based on accuracy maximization. Both C4.5 and C5.0 have rule set generators that are a commonly used alternative to pruning [66]. Given a decision tree, these methods produce a set of rules that is typically simpler and that generalizes better to new examples than the original tree. However, these methods are also based on accuracy maximization, so they are also unsuitable for probability estimation.

We show how to improve directly the accuracy of decision tree probability estimates. Our experiments use C4.5 without pruning and without collapsing to obtain raw scores that can be transformed into accurate class membership probabilities. The choice to do no pruning is supported by the results of Bradford *et al.* [12], who find that performing no pruning and variants of pruning adapted to loss minimization both lead to similar performance. Not using pruning is also suggested by Bauer and Kohavi [4] in their Section 7.3.

The methods we propose transform the leaf scores of a standard decision tree. Completely different methods have been suggested, but they have major drawbacks. Smyth *et al.* [73] use kernel density estimators at the leaves of a

decision tree. However their algorithms are based on C4.5 and CART with pruning, so they are unsuitable for highly unbalanced datasets. Their experiments use only synthetic, reasonably balanced datasets. Our experiments use an unbalanced real-world dataset where the less probable class has a base rate of only about 5%. Estimating probabilities using bagging has been suggested by Breiman [14] and by Domingos [20], but as explained above in Section III.A, bagging does not give unbiased probability estimates in general.

III.C.2 Smoothing

One way of improving the probability estimates given by a decision tree is to make these estimates smoother, i.e. to adjust them to be less extreme. Provost and Domingos [64] suggest using the Laplace correction method. For a two-class problem, this method replaces the conditional probability estimate $p = \frac{k}{n}$ by

$$p' = \frac{k+1}{n+2}.$$

The Laplace correction method adjusts probability estimates to be closer to 1/2, which is not reasonable when the two classes are far from equiprobable, as is the case in many real-world applications.

In general, one should consider the overall average probability of the positive class, that is, the base rate, when smoothing probability estimates.

We replace the probability estimate $p = \frac{k}{n}$ by

$$p' = \frac{k + b \cdot m}{n + m},$$

where b is the base rate of the positive class and m is a parameter that controls how much scores are shifted towards b. This smoothing method is called m-estimation [17]. For example, if a leaf contains four training examples, one of which is positive, the raw C4.5 decision tree score of any example assigned to this leaf is 0.25. The smoothed score with $m = 200$ and $b = 0.05$ is

$$p' = \frac{1 + 0.05 \cdot 200}{4 + 200} = \frac{11}{204} = 0.0539.$$

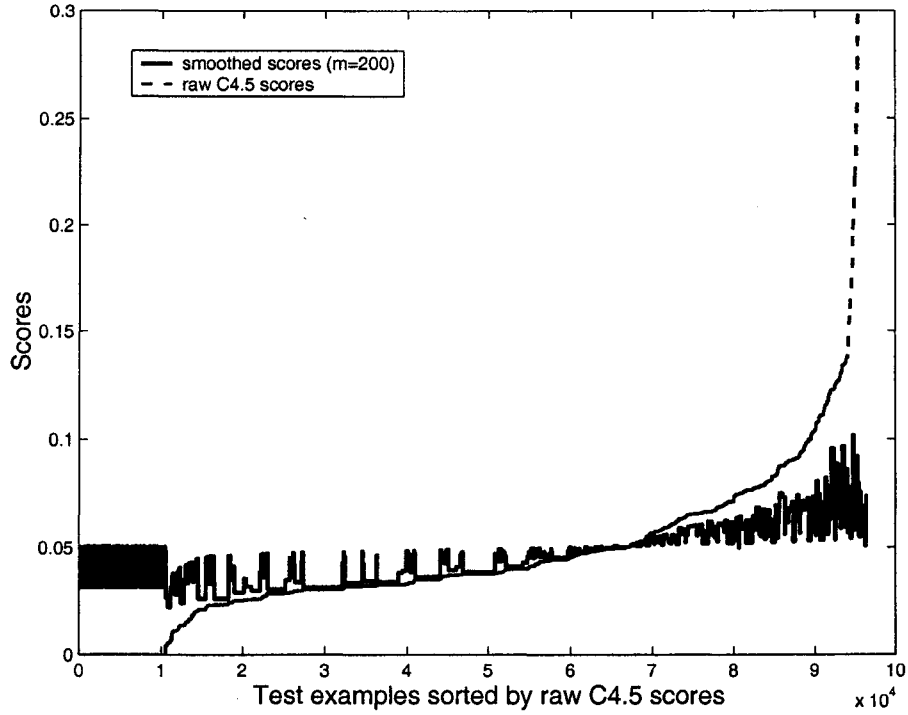


Figure III.C.1: Smoothed scores and raw C4.5 scores for test examples sorted by raw score. The figure shows how the scores change after smoothing is applied. In particular, examples that are assigned a score close to 0 (left-hand side) or 1 (right-hand side) by C4.5 have their scores significantly shifted towards the base rate by smoothing.

As m increases, observed training set frequencies are shifted more towards the base rate.

Previous research has suggested choosing m by cross-validation. Given a base rate b, we suggest using m such that $bm = 10$ approximately. This heuristic ensures that raw probability estimates that are likely to have high variance, those with $k \leq 10$, are given low credence. Experiments show that the effect of smoothing by m-estimation is qualitatively similar for a wide range of values of m, so, as is highly desirable, the precise value chosen for m is unimportant.

Figure III.C.1 shows the smoothed scores with $m = 200$ of the KDD-98 test set examples sorted by their raw C4.5 scores. As expected, smoothing shifts all

scores towards the base rate of approximately 0.05, which is desirable given that C4.5 scores tend to be overestimates or underestimates. While raw C4.5 scores range from 0 to 1, smoothed scores range from 0.0224 to 0.1018.

III.C.3 Curtailment

As discussed above, without pruning decision tree learning methods tend to overfit training data and to create leaves in which the number of examples is too small to induce conditional probability estimates that are statistically reliable (which we call small leaves). Smoothing attempts to correct these estimates by shifting them towards the overall average probability, i.e. the base rate b. However, if the parent of a small leaf contains enough examples to induce a statistically reliable probability estimate, then assigning this estimate to a test example associated with the leaf may be more accurate then assigning it a combination of the base rate and the observed leaf frequency, as done by smoothing. If the parent of a small leaf still contains too few examples, we can use the score of the grandparent of the leaf, and so on until the root of the tree is reached. At the root, of course, the observed frequency is the training set base rate.

We call this method of improving conditional probability estimates curtailment because when classifying an example, we curtail search through the decision tree as soon as we reach a node that has less than v examples, where v is a parameter of the method. The score of the parent of this node is then assigned to the example in question. As for smoothing, v can be chosen by cross-validation, or using a heuristic such as making $bv = 10$. We choose $v = 200$ for all our experiments. Informal experiments show that values of v between 100 and 400 give similar results, so the exact setting of v is not critical.

Given the KDD-98 training set, curtailment effectively creates the decision tree shown in part in Figure III.C.2. The distinction between internal nodes and leaves is blurred in this tree, because a node may serve as an internal node for some examples and as a leaf for others, depending on the attribute values of

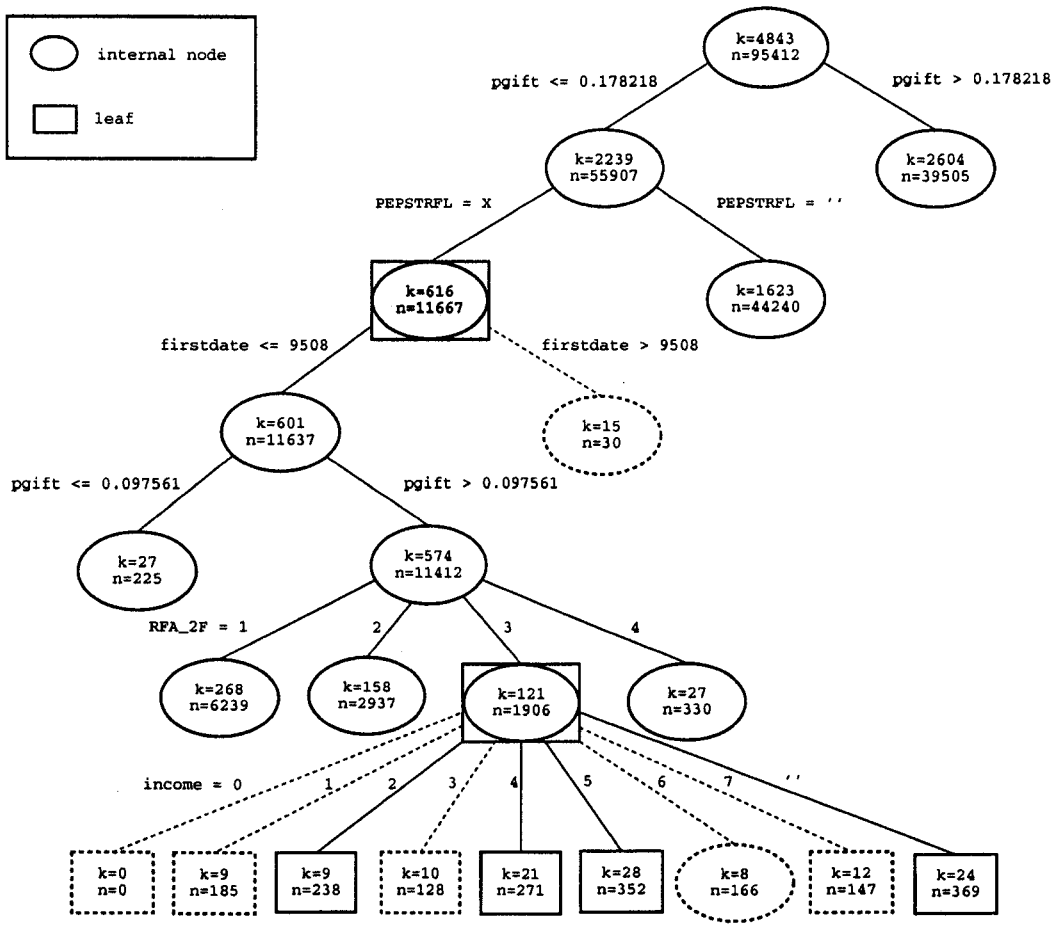


Figure III.C.2: Part of the decision tree obtained by curtailment with $v = 200$. The dotted nodes are present in the original C4.5 tree, but are effectively eliminated from the curtailment tree because $n < v$.

the examples. The node in gray is an example of a node that can serve both as an internal node and as a leaf, because one of its branches has been eliminated from the tree, but not all.

Curtailment is not equivalent to any type of pruning, nor to traditional early stopping during the growing of a tree, because those methods eliminate all the children of a node simultaneously. In contrast, curtailment may eliminate some children and keep others, depending on the number of training examples associated with each child. Intuitively, curtailment is preferable to pruning for

probability estimation because nodes are removed from a decision tree only if they are likely to give unreliable probability estimates.

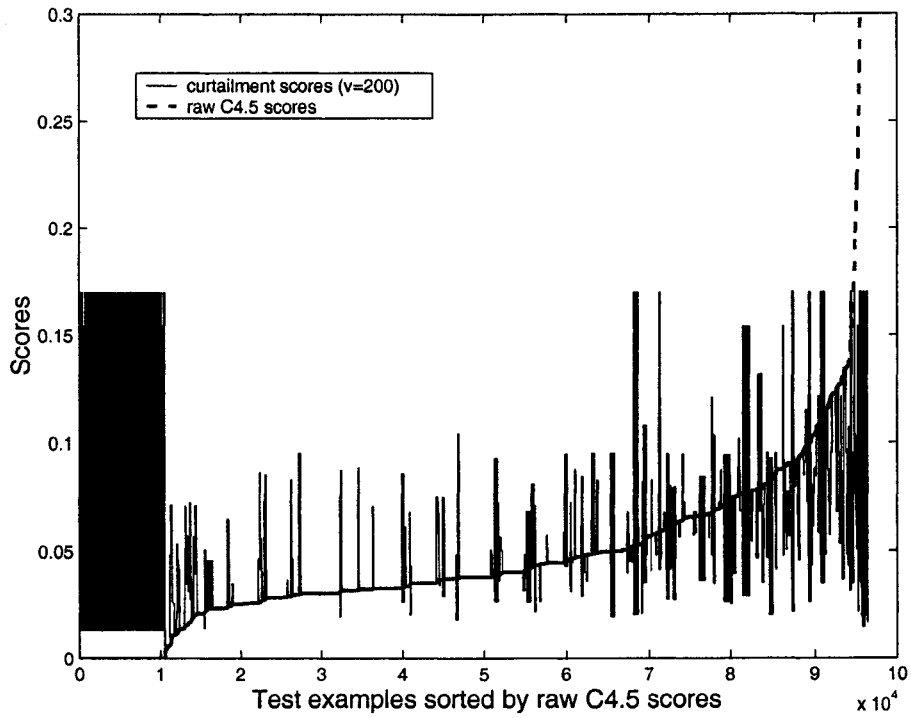


Figure III.C.3: Curtailment scores and raw C4.5 scores for test examples. Examples are sorted by raw C4.5 score. The figure shows that scores change significantly after curtailment is applied, in particular for examples that are assigned a score close to 0 (left-hand side) or 1 (right-hand side) by C4.5.

Figure III.C.3 shows the curtailment scores with $v = 200$ of the KDD-98 test set examples sorted by their raw C4.5 scores. The jagged lines in the chart show that many scores are changed significantly by curtailment. Overall, the range of scores is reduced as with smoothing, but not as much. The minimum curtailment score is 0.0045 while the maximum is 0.1699.

III.C.4 Binning

Naive Bayesian classifiers are based on the assumption that within each class, the values of the attributes of examples are independent.

Mathematically, this conditional independence assumption can be stated as

$$P(x|j) = \prod_{k=1}^{n} P(x_k|j)$$

where each x_k is one attribute value of x.

It is well-known that these classifiers tend to give inaccurate probability estimates [21]. Given an example x, suppose that a naive Bayesian classifier computes the score $n(x)$. Because attributes tend to be positively correlated, these scores are typically too extreme: for most x, either $n(x)$ is near 0 and then $n(x) < P(j = 1|x)$ or $n(x)$ is near 1 and then $n(x) > P(j = 1|x)$. However, naive Bayesian classifiers tend to rank examples well: if $n(x) < n(y)$ then $P(j = 1|x) < P(j = 1|y)$.

We use a histogram method to obtain calibrated probability estimates from a naive Bayesian classifier. We sort the training examples according to their scores and divide the sorted set into b subsets of equal size, called bins. For each bin we compute lower and upper boundary $n(\cdot)$ scores. Given a test example x, we place it in a bin according to its score $n(x)$. We then estimate the corrected probability that x belongs to class j as the fraction of training examples in the bin that actually belong to j.

The number of different probability estimates that binning can yield is limited by the number of alternative bins. This number, $b = 10$ in our experiments, must be small in order to reduce the variance of the binned probability estimates, by increasing the number of examples whose 0/1 memberships are averaged inside each bin. Binning reduces the resolution, i.e. the degree of detail, of conditional probability estimates, while improving the accuracy of these estimates by reducing both variance and bias compared to uncalibrated estimates.

Binning is a discrete non-parametric method for calibrating probability estimates. In future work, we should consider using continuous methods such as the super-smoother or loess to obtain calibrated probability estimates with a greater degree of detail. Sobehart *et al.* [74] use a Gaussian kernel regression method in a similar context. Applying parametric methods to calibrate naive Bayes scores is not straightforward. For example, Bennett [8] reports that sigmoid functions cannot transform naive Bayes scores into well-calibrated probability estimates.

With most learning methods, in order to obtain binned estimates that do not overfit the training data, we should partition the training set into two subsets. One subset would be used to learn the classifier that yields uncalibrated scores, while the other subset would be used for the binning process. More training examples would be assigned to the first subset because learning a classifier involves setting many more parameters than setting the binned probabilities. For naive Bayesian classifiers, however, separate subsets are not necessary because this learning method does not overfit the training data much. So we use the entire training set both for learning the classifier and for binning.

III.C.5 Averaging probability estimates

If different methods provide noisy probability estimates that are partially uncorrelated, it is intuitive that averaging the probability estimates given by these methods reduces the noise, thereby improving the probability estimates.

This intuition is formalized by Tumer and Ghosh [83]. They show that by combining the probability estimates given by different classifiers through averaging we can reduce the variance of the probability estimates. The reduction in the variance depends on the degree of correlation of the noise in the probability estimates produced by each classifier and on how many classifiers are used.

Assuming that the variance of the probability estimates given by each classifier is approximately the same, the variance of the averaging combiner is

given by

$$\bar{\sigma}^2 = \frac{1+\rho(N-1)}{N}\sigma^2$$

where σ^2 is the variance of each original classifier, N is the number of classifiers and ρ is the correlation factor among all classifiers. If the classifiers are independent ($\rho = 0$), the combined variance is reduced by N. On the other hand, if the classifiers are completely correlated ($\rho = 1$), the variance is unchanged.

Since the probability estimates obtained from the decision tree and naive Bayesian classifiers are partially uncorrelated, averaging them should yield estimates that are more accurate than those given by each individual method. In Section III.E we show experimental results that confirm this hypothesis.

III.D Estimating donation amounts

In general, in cost-sensitive learning we need to estimate example-specific misclassification costs, in addition to example-specific class conditional probabilities. We need to estimate misclassification costs for training examples when using MetaCost, and for test examples when using direct cost-sensitive decision-making.

If costs and probabilities are both unknown, then estimating costs well can be more important for making good decisions than estimating probabilities well. Cost estimates are more important if the relative variation of costs across different examples is greater than the relative variation of probabilities. The dynamic range of costs may be greater than the dynamic range of probabilities either because the dynamic range of true costs is greater, or because estimating costs accurately is easier than estimating probabilities accurately.

In the KDD-98 domain for example, estimating donation probabilities is difficult. Our best method for this task, the averaging of smoothing, curtailment, and binned naive Bayes, gives conditional probabilities in the narrow range from 0.0172 to 0.1189. Estimating donation amounts is easier because past amounts are excellent predictors of future amounts.

It may appear that for non-donors in the training set we should impute a donation amount of zero, since their actual donation amount is zero. But this imputation would be analogous to imputing a donation probability of zero for the non-donors based on the fact that they have not donated, which is clearly wrong. When responding to a solicitation a person has to make two decisions. The first is whether to donate or not, while the second is how much to donate. Conceptually, these decisions are governed by two different random processes, not necessarily sequential or independent of course. For donors in the training set, the outcome of the random process that sets the donation amount is known, while for non-donors, this outcome is unknown. For individuals in the test set, the outcome of both random processes is unknown. Whenever the outcome of one or both processes is unknown, the learning task is to estimate its outcome. For non-donors in the training set, the task is to estimate the amounts that they would have donated, if they had made donations.

It is also wrong to impute any fixed quantity as a donation estimate for test examples. Using the same donation estimate for all test examples means that the decision whether or not to solicit a person is based exclusively on the probability that they will donate. This method is equivalent to using a fixed cost matrix for test examples.

In general, whenever misclassification costs are assumed to be fixed, different decisions for different examples can only be based on different conditional probability estimates for those examples.

For clarity, the arguments in the previous paragraphs are expressed in language that is specific to the donations domain. However, similar arguments apply to any scenario where costs or benefits are different for different examples. These costs or benefits must be estimated for each example, whenever they are unknown. Assuming that unknown costs or benefits are zero or constant is incorrect.

The method we use for estimating donation amounts is least-squares multiple linear regression (MLR). The donors in the training set that have donated

at most $50 are used as input for the regression, which is based on one original attribute and one derived attribute:

- `lastgift`: dollar amount of most recent gift,
- `ampergift`: average gift amount in responses to the last 22 promotions.

Since the topic of this research is not variable selection, we somewhat arbitrarily choose these two attributes based on previous work. We use the linear regression equation to estimate donation amounts for all examples in both the training and test sets.

Donations of more than $50 are very rare in our domain: 46 of 4843 donations recorded in the training set. We eliminate these examples from the regression training set as a heuristic attempt to reduce the impact of outliers on the regression. If included, these examples have the most influence on the regression equation, because they have the highest y values and the regression equation is chosen to minimize the sum of squared y errors. However, it is less important to estimate y values accurately for these individuals, because the optimal decision is always to solicit them, given that predicted donation probabilities are always over 1.5%. Accurate predicted donation probabilities are never close to zero because of the intrinsic difficulty of predicting whether or not a person will donate. In future work, we shall consider using non-linear regression methods that are able to cope adaptively with outliers.

Conversely, making accurate predictions is most important for individuals whose expected donation is close to $0.68. These individuals all have estimated donation amounts under $50, again because all predicted donation probabilities are over 1.5%.

III.D.1 The problem of sample selection bias

When estimating donation amounts, a fundamental problem is that any estimator, for example a regression equation, must be learned based on examples of

Prob. estimation method	Without Heckman		With Heckman	
	Training set	Test set	Training set	Test set
Smoothed C4.5 (sm)	$19256	$14093	$18583	$14321
C4.5 with curtailment (cur)	$16722	$13670	$17037	$14161
Binned naive Bayes (binb)	$14262	$14208	$14994	$15094
Average(sm, cur)	$18591	$14518	$18474	$14879
Average(sm, cur, binb)	$18140	$14877	$17400	$15329

Table III.3: Profit attained on the training and test sets using each probability estimation method.

people who actually donate. But this estimator must then be applied to a different population, i.e. both donors and non-donors. This problem is known in general as sample selection bias. It occurs whenever the training examples used to learn a model are drawn from a different probability distribution than the examples to which the model is applied.

In the donations domain, the donation amount and the probability of donation are negatively correlated. People who are more likely to respond to a solicitation tend to make smaller donations, while people who make larger donations are less likely to respond. This relationship is illustrated in Figure III.D.1. Since examples of people who actually donate are the only training examples for the regression, donation amounts estimated by the regression equation tend to be too low for test examples that have a low probability of donation.

As we have explained previously [27], the standard method of compensating for sample selection bias in econometrics is a two-step procedure due to James J. Heckman of the University of Chicago [43]. In October 2000 Heckman was awarded the Nobel prize in economics for developing and applying this procedure. Expressed using our notation, Heckman's procedure is applicable when each example x belongs to one of two classes, i.e. $j(x) = 0$ or $j(x) = 1$, and the dependent variable to be estimated $y(x)$ is observed for a training example if and only if $j(x) = 1$. The first step of the procedure is to learn a probit linear model to estimate conditional probabilities $P(j = 1|x)$. A probit model is a variant of

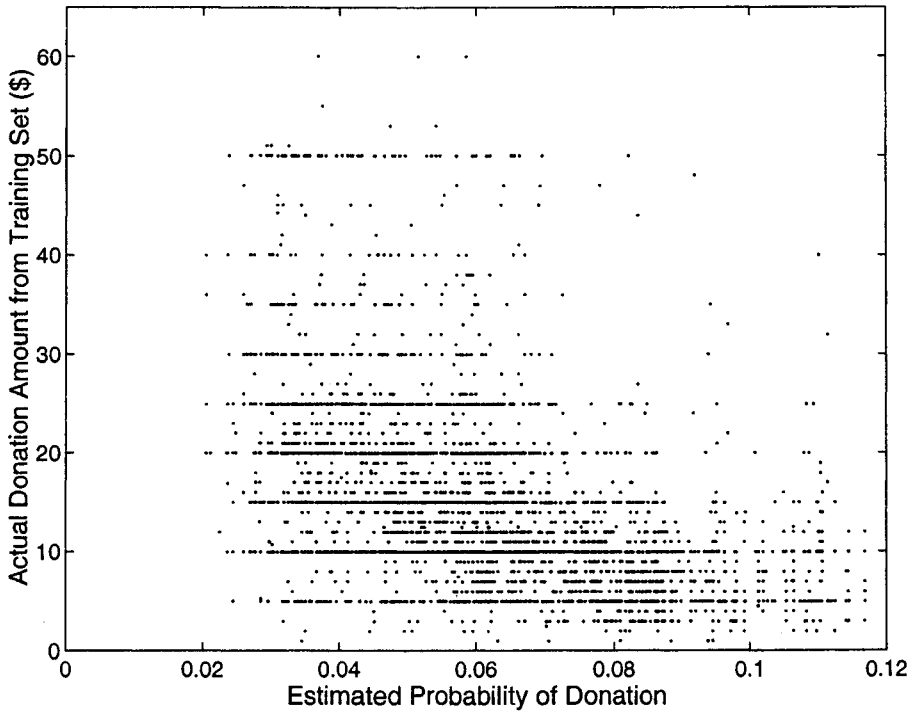


Figure III.D.1: Actual donation amount versus estimated probability of donation, for all donors in the training set. A negative correlation between donation amount and probability of donation is visible.

logistic regression where the cumulative Gaussian probability density function is the sigmoid function. The second step of Heckman's procedure is to estimate $y(x)$ by linear regression using only the training examples x for which $j(x) = 1$, but including for each x a transformation of the estimated value of $P(j = 1|x)$. Heckman has proved that this procedure yields estimates of $y(x)$ that are unbiased for all x, regardless of whether $j(x) = 0$ or $j(x) = 1$, under certain conditions [43].

Our second method for estimating donation amounts is a nonlinear variant of Heckman's procedure. Instead of using a linear estimator for $P(j = 1|x)$, we use a decision tree or a naive Bayes classifier to obtain probability estimates, as described in Section III.C. We then include these probability estimates directly as an additional attribute when applying a learning method to obtain an estimator for $y(x)$. This learning method could be a nonlinear method, for example a neural

network method, but in order to investigate carefully the usefulness of Heckman's idea, we hold everything else constant and just provide the estimated $P(j = 1|x)$ values as a third attribute of x to a linear regression that is otherwise the same as in the first method.

III.E Experimental Results

In this section, we investigate experimentally how the new probability and cost estimation methods described above affect the profit attained on the KDD-98 dataset (described in Chapter II, Section II.C). We first report our results, and then discuss the issue of statistical significance.

For each of the probability estimation methods described in Section III.C, Table III.3 shows the profit obtained when we use the multiple linear regression that includes only `lastgift` and `ampergift` as attributes, and when we apply Heckman's procedure by including the probability estimates as an additional attribute to the regression. When we use Heckman's procedure, the profit on the test set increases for all probability estimation methods, on average by $484. The fact that the improvement is systematic indicates that Heckman's procedure succeeds in correcting sample selection bias.

To implement MetaCost, probability and donation estimates obtained as described in Sections III.C and III.D are used to relabel the training set according to Equation III.A.1. We train C4.5, with pruning and collapsing, on the relabeled training examples and apply the resulting decision tree to the training and test examples. The profit obtained from mailing the people who are labeled positive by the decision tree is given in Table III.4.

Comparing the results in Table III.4 with the results in the second half of Table III.3, we see that MetaCost performs consistently less well than direct cost-sensitive decision-making. On average, the profit achieved with MetaCost on the test set is $1751 lower than the profit achieved with direct cost-sensitive

Probability estimation method	Training set	Test set
Smoothed C4.5 (sm)	\$17359	\$12835
C4.5 with curtailment (cur)	\$15869	\$11283
Binned naive Bayes (binb)	\$13608	\$14113
Average(sm, cur)	\$17547	\$13284
Average(sm, cur, binb)	\$16531	\$13515

Table III.4: Profit attained on the training and test sets using MetaCost with each probability estimation method. Donation amount estimates are obtained from the MLR with the Heckman adjustment.

decision-making. The best result with MetaCost is \$14113, while the best result with the direct method is \$15329, which is better than the result obtained by the winner of the KDD-98 contest, \$14712.

We conclude that direct cost-sensitive decision-making is preferable to MetaCost. We attribute the worse performance of MetaCost to the difficulty that any single model must have in estimating costs and probabilities as accurately as two separate models. Learning a single classifier from relabeled training data causes more errors in approximating the ideal decision boundary than learning two estimators.

It is difficult to make definite statements about the statistical significance of the experimental results above. There are 4872 donors in the fixed test set. For these individuals, the average donation is \$15.62. On a different test set drawn randomly from the same probability distribution, one would expect a one standard deviation fluctuation up or down of $\sqrt{4872}$ in the number of donors. This fluctuation would cause a change of about $\$15.62 \cdot \sqrt{4872} = \$1090$ in total profit. Therefore, a profit difference of less than \$1090 between two methods is not statistically significant.

Many of the profit differences between methods that we observe are less than \$1090. There are several avenues we could follow to obtain statistically significant differences between methods. One avenue would be to use cross-validation,

instead of a single training set and a single test set. However, the training set/test set split we use is standard. If we did not use it, our results would not be comparable with those of previous work using the same dataset.

Another avenue would be to use multiple datasets for comparing different methods, as done for example by Domingos [20]. But, despite the unquestioned importance of example-dependent costs in many learning tasks, the KDD-98 dataset is the only dataset in the UCI repositories for which real-world misclassification cost information is available. Most previous experimental research on cost-sensitive learning has used arbitrary cost matrices. We prefer to use real cost data, especially since we are interested in the situation where costs are different for different examples.

The purpose of the experiments reported here is not so much to identify a single best method for cost-sensitive learning and decision-making, but rather to compare the usefulness of the alternative sub-methods proposed in previous sections. In all trials, the test set profit achieved using MetaCost is lower and using Heckman's procedure is higher. We choose not to quantify the level of this statistical significance because doing so would require making assumptions that are certainly false, and therefore give misleading conclusions. In particular, because all trials use the same training and test sets, they are not statistically independent. Always using the same training and test set removes one source of variance, so even small differences in performance between data mining methods are in fact likely to be genuine [55].

III.F Conclusions

The main contributions of this chapter are the following:

1. We explain a general method of cost-sensitive learning that performs systematically better than MetaCost in our experiments.

2. We provide the first general solution to the fundamental problem of costs

being different for different examples, and unknown for some of the examples.

3. As part of (2), we identify and solve the problem of sample selection bias, i.e. the fact that the training set available for learning to estimate costs is not representative of test examples, or indeed of other training examples. In Chapter VI we will tackle the problem of sample selection bias in a more general setting.

All the methods we propose are evaluated carefully with experiments using a large, difficult and highly cost-sensitive real-world dataset. Previous research has tended to use small datasets with synthetic cost data.

We have used simple methods for both probability estimation and cost estimation in this chapter in order to illustrate our general cost-sensitive learning approach and to provide a baseline for future research. In chapter V we present improved methods obtaining calibrated probability estimates from classifiers.

Our experiments are designed so that both MetaCost and the alternative we propose use the same methods for estimating costs and probabilities. Therefore, we expect our conclusion that direct cost-sensitive decision-making is preferable to remain valid with other estimation methods. In particular, both MetaCost and direct cost-sensitive decision-making will be improved by any improvement in techniques for probability estimation.

Acknowledgments

The text of this chapter, in part, is a reprint of the material as it appears in the *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* [95]. The dissertation author was the primary author, and the co-author listed in this publication directed and supervised the research which forms the basis for the chapter.

Chapter IV

Cost-sensitive learning by example weighting

In this chapter, we present a family of methods for cost-sensitive learning that uses the importance formulation and is motivated by a folk theorem that we formalize and prove in section IV.A. This theorem states that altering the original distribution of training examples D to another $\hat{D}$ by weighting each example proportionately to its relative cost (or importance) makes any error-minimizing classifier learner accomplish expected cost minimization on the original distribution. Representing samples drawn from $\hat{D}$, however, is more challenging than it may seem. There are two basic methods for doing this, starting with a set of examples drawn from D:

- Transparent Box (Section IV.B): Supply the example-dependent costs as example weights to the classifier learning algorithm.
- Black Box (Section IV.C): Carefully resample using these same weights.

While the transparent box approach cannot be applied with arbitrary classifier learners, it can be applied to many, including any classifier which only uses the data to calculate expectations. We show empirically in Section IV.D.1 that this method results in good performance. The black box approach has the

advantage that it can be applied to any classifier learner. It turns out, however, that simple resampling-with-replacement can result in severe overfitting related to duplicate examples), as is confirmed by our experimental results in Section IV.D.2.

We propose, instead, to employ *cost-proportionate rejection sampling* to realize the latter approach, which allows us to independently draw examples according to $\hat{D}$. (In essence, this method accepts each example in the input sample with probability proportional to its associated weight.) This method comes with a theoretical guarantee: In the worst case this sampling method produces a classifier that achieves at least as good approximate cost minimization as applying the base classifier learning algorithm on the entire sample. This is a remarkable property for a subsampling scheme: in general, we expect any technique using only a subset of the examples to compromise predictive performance.

The runtime savings made possible by this sampling technique enable us to run the classification algorithm on multiple draws of subsamples and average over the resulting classifiers. This last method is what we call *costing* (cost-proportionate rejection sampling with aggregation). Costing allows us to use an arbitrary cost-insensitive learning algorithm as a black box in order to accomplish cost-sensitive learning, achieves excellent predictive performance and can achieve drastic savings of computational resources, both in terms of time and space.

IV.A A Folk Theorem

In the importance formulation (see Chapter II, section II.A.2) we assume that examples are drawn independently from a distribution D with domain $X \times Y \times C$ where X is the input space to a classifier, Y is a (binary) output space and $C \subset [0, \infty)$ is the importance (extra cost) associated with mislabeling that example. The goal is to learn a classifier $h : X \to Y$ which minimizes the expected cost,

$$E_{x,y,c\sim D}[c\, I(h(x) \neq y)]$$

given training data of the form: (x, y, c), where $I(\cdot)$ is the indicator function that has value 1 in case its argument is true and 0 otherwise.

A basic folk theorem states that if we have examples drawn from the distribution:

$$\hat{D}(x, y, c) \equiv \frac{c}{E_{x,y,c\sim D}[c]} D(x, y, c)$$

then optimal error rate classifiers for $\hat{D}$ are optimal cost minimizers for data drawn from D. We say "folk theorem" here because the result appears to be known by some and it is straightforward to derive it from results in decision theory, although we have not found it published.

Theorem IV.A.1. *(Translation Theorem) For all distributions, D, there exists a constant $N = E_{x,y,c\sim D}[c]$ such that for all classifiers, h:*

$$E_{x,y,c\sim \hat{D}}[I(h(x) \neq y)] = \frac{1}{N} E_{x,y,c\sim D}[c\, I(h(x) \neq y)]$$

Proof.

$$\begin{aligned} E_{x,y,c\sim D}[c\, I(h(x) \neq y)] &= \sum_{x,y,c} D(x, y, c) c\, I(h(x) \neq y) \\ &= N \sum_{x,y,c} \hat{D}(x, y, c) I(h(x) \neq y) \\ &= N E_{x,y,c\sim \hat{D}}[I(h(x) \neq y)] \\ \text{where} \quad & \hat{D}(x, y, c) = \frac{c}{N} D(x, y, c). \end{aligned}$$

□

Despite its simplicity, the translation theorem is useful to us because the right-hand side expresses the expectation we want to control (via the choice of h) and the left-hand side is the probability that h errs under another distribution. Choosing an h to minimize the rate of errors under $\hat{D}$ is equivalent to choosing a h to minimize the expected cost under D. Similarly, ϵ-approximate error rate minimization under $\hat{D}$ is equivalent to $N\epsilon$-approximate expected cost minimization under D.

The prescription for coping with cost-sensitive problems is straightforward: reweight the distribution in your training set according to the importances so that the training set is effectively drawn from $\hat{D}$. Doing this in a correct and general manner is more challenging than it may seem and is the topic of the rest of the chapter.

IV.B Transparent Box: Using Weights Directly

IV.B.1 General conversion

Here we examine how importance weights can be used directly. The approach taken here is a transparent box approach where access to the source code is required, and *not* a black box approach (which we develop in the next section). In particular, we use the weights within the learning algorithm to accomplish cost-sensitive classification.

The mechanisms for realizing the transparent box approach have been described elsewhere for a number of *weak learners* used in boosting, but we will describe them here for completeness.

The classifier learning algorithm must use the weights so that it effectively learns from data drawn according to $\hat{D}$. This specific requirement is easy to apply for all learning algorithms which fit the statistical query model [46].

As shown in figure IV.B.1, many learning algorithms can be divided into two components: a portion which calculates the (approximate) expected value of some function (or query), say f, and "the rest"— a portion which forms these queries and uses their output to construct a classifier. For example, neural networks (with batch-mode gradient updates), decision trees, and Naive Bayes classifiers can be constructed in this manner. Support vector machines are not easily constructible in this way, because the individual classifier is explicitly dependent upon individual examples (rather than on statistics derived from the sample set).

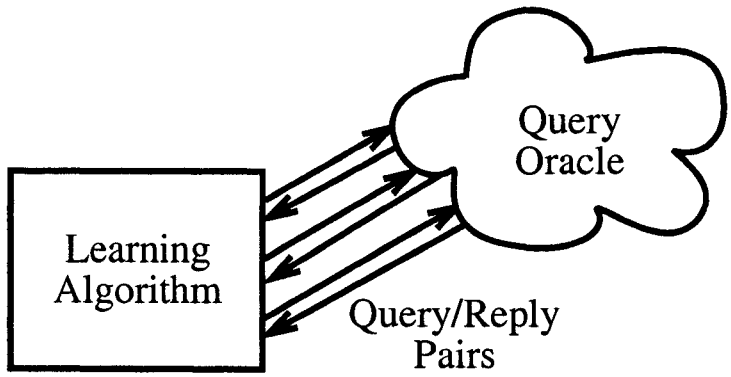


Figure IV.B.1: A figure showing the statistical query model of learning algorithms. For any learning algorithm decomposable in this form, there is a generic method for learning from a reweighted distribution directly.

With finite data we cannot precisely calculate

$$E_{x,y\sim D}[f(x,y)].$$

However, with high probability we can approximate this expectation, given a set of samples drawn independently from the underlying distribution D.

Whenever we have a learning algorithm that can be decomposed as in figure IV.B.1, there is a simple recipe for using the weights directly. Instead of simulating the expectation with

$$\frac{1}{|S|}\sum_{(x,y)\in S} f(x,y),$$

we use

$$\frac{1}{\sum_{(x,y,c)\in S} c}\sum_{(x,y,c)\in S} cf(x,y).$$

This method is equivalent to importance sampling for $\hat{D}$ using the distribution D, and so the modified expectation is an unbiased Monte Carlo estimate of the expectation w.r.t. $\hat{D}$.

Note that even when a learning algorithm does not have a statistical query form, it may be possible to incorporate importance weights directly. We

now discuss how to incorporate importance weights into some specific learning algorithms.

IV.B.2 Naive Bayes and boosting

Naive Bayes learns by calculating empirical probabilities for each output y using Bayes' rule and assuming that each feature is independent given the output:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{\prod_i P(x_i|y)P(y)}{\prod_i P(x_i)}$$

Each probability estimate in the above expression can be thought of as a function of empirical expectations according to D, and thus it can be formulated in the statistical query model. For example, $p(x_i|y)$ is just the expectation of

$$I(\mathbf{x_i} = x_i) \wedge I(\mathbf{y} = y)$$

divided by the expectation of $I(\mathbf{y} = y)$.

More specifically, to compute the empirical estimate of $P(x_i|y)$ with respect to D, we need to count the number of training examples that have y as output, and those having x_i as the i-th input dimension among those. When we compute these empirical estimates with respect to $\hat{D}$, we simply have to sum the weight of each example, instead of counting the examples. (This property is used in the implementation of boosted Naive Bayes [26].)

To incorporate importance weights into AdaBoost [33], we give the importance weights to the weak learner in the first iteration, thus effectively drawing examples from $\hat{D}$. In the subsequent iterations, we use the standard AdaBoost rule to update the weights. Therefore, the weights are adjusted according to the accuracy on $\hat{D}$, which corresponds to the expected cost on D.

Note that AdaCost [31], a variant of AdaBoost for cost-sensitive learning, has also been proposed. AdaCost uses a modified update rule to incorporate costs and improved performance is observed. In contrast, Proposition IV.A.1 implies that such a modification is not necessary if we start with examples drawn

from $\hat{D}$. This may seem contradictory, but note that Proposition IV.A.1 is purely about error translation and *not* about learning algorithm design. From the viewpoint of the proposition, AdaCost is a learning algorithm with a different bias than the AdaBoost bias. Just as other boosting algorithms such as LogitBoost [34] are sometimes superior to AdaBoost, AdaCost may be sometimes superior to AdaBoost.

IV.B.3 C4.5

C4.5 [66] is a widely used decision tree learner. There is a standard way of incorporating example weights to C4.5, which in the original algorithm was intended to handle missing attributes (examples with missing attributes were divided into fractional examples, each with a smaller weight, during the growth of the tree). This same facility was later used by Quinlan in the implementation of boosted C4.5 [67].

IV.B.4 Support Vector Machine

In its basic form, the SVM algorithm [86, 44] learns the parameters a and b describing a linear decision rule

$$h(x) = \text{sign}(a \cdot x + b),$$

so that the smallest distance between each training example and the decision boundary (called the margin) is maximized. It works by solving the following optimization problem:

$$\begin{aligned} &\text{minimize: } V(a, b, \xi) = \tfrac{1}{2} a \cdot a + C \textstyle\sum_{i=1}^{n} \xi_i \\ &\text{subject to: } \forall i: \quad y_i[a \cdot x_i + b] \geq 1 - \xi_i, \ \xi_i > 0 \end{aligned}$$

The constraints require that all examples in the training set are classified correctly up to some slack ξ_i. If a training example lies on the wrong side of the decision boundary, the corresponding ξ_i is greater than 1. Therefore, $\sum_{i=1}^{n} \xi_i$ is an upper

bound on the number of training errors. The factor C is a parameter that allows one to trade off training error and model complexity. The algorithm can be generalized to non-linear decision rules by replacing inner products with a kernel function [86] in the formulas above.

The SVM algorithm does *not* have the form of a statistical query algorithm. Despite this drawback, it is possible to incorporate importance weights in a heuristic way. First, we note that $\sum_{i=1}^{n} c_i \xi_i$, where c_i is the importance of example i, is an upper bound on the total cost. Therefore, we can modify $V(a, b, \xi)$ to

$$V(a, b, \xi) = \tfrac{1}{2} a \cdot a + C \sum_{i=1}^{n} c_i \xi_i.$$

Now C controls the trade off model complexity and total cost.

The SVMLight package [45] allows users to input importance weights c_i and works with the modified $V(a, b, \xi)$ as above, although this feature has not yet been documented.

IV.C Black Box: Sampling methods

Now suppose we do not have transparent box access to the learner. In this case, sampling is the obvious method to use in converting from one distribution of examples to another to obtain a cost-sensitive learner using the translation theorem (Proposition IV.A.1). As it turns out, straightforward sampling methods do not work well in this case, motivating us to propose an alternative method based on rejection sampling.

IV.C.1 Resampling

Resampling-with-replacement is a sampling scheme where each sample (x, y, c) is drawn according to the distribution

$$p(x, y, c) = \frac{c}{\sum_{(x,y,c) \in S} c}.$$

Many samples are drawn to create a resampled dataset S'. This method, at first pass, appears useful because every sample is effectively drawn from the distribution $\hat{D}$. In fact, very poor performance can result when using this technique, which is essentially due to overfitting because of the fact that the samples in S' are not drawn *independently* from $\hat{D}$, as we will elaborate in the section on experimental results (Section IV.D.)

Resampling-without-replacement is also not a solution to this problem. In resampling-without-replacement, a sample, (x, y, c) is drawn from the distribution

$$p(x, y, c) = \frac{c}{\sum_{(x,y,c)\in S} c}$$

and the next sample is drawn from the set $S - \{x, y, c\}$. This process is repeated, drawing from a smaller and smaller set according to the weights of the samples remaining in the set.

To see how this method fails, note that resampling-without-replacement m times from a set of size m results in the original set, which (by assumption) is drawn from the distribution D, and not $\hat{D}$ as desired.

IV.C.2 Cost-proportionate rejection sampling

There is another sampling scheme called rejection sampling [87] which allows us to draw samples independently from the distribution $\hat{D}$, given samples drawn independently from D, and thus avoids the duplication problem. In rejection sampling, samples from $\hat{D}$ are drawn by first drawing samples from D, and then keeping the sample with probability proportional to $\hat{D}/D$. Here, we have $\hat{D}/D \propto c$, so we accept an example with probability c/Z, where Z is some constant chosen so that

$$\max_{(x,y,c)\in S} c \leq Z,$$

leading to the name *cost-proportionate rejection sampling.*

In practice, we choose the minimal

$$Z = \max_{(x,y,w)\in S} c$$

so as to maximize the size of the resampled set S'. A data-dependent choice of Z is *not* formally allowed for rejection sampling. However, the introduced bias appears small when $|S| >> 1$. A precise measurement of "small" is an interesting theoretical problem.

Rejection sampling results in a set S' which is generally smaller than S. Furthermore, because inclusion of a sample in S' is independent of other samples, and the samples in S are drawn independently, we know that the samples in S' are distributed independently according to $\hat{D}$.

Using cost-proportionate rejection sampling to create a set S' and then using a learning algorithm $A(S')$ is guaranteed to produce an approximately cost-minimizing classifier, as long as the learning algorithm A achieves approximate minimization of classification error.

Theorem IV.C.1. *(Correctness) For all cost-sensitive sample sets S, if cost-proportionate rejection sampling produces a sample set S' and $A(S')$ achieves ϵ classification error:*

$$E_{x,y,c\sim\hat{D}}[I(h(x) \neq y)] \leq \epsilon$$

then $h = A(S')$ approximately minimizes cost:

$$E_{x,y,c\sim D}[c\, I(h(x) \neq y)] \leq \epsilon N$$

where $N = E_{x,y,c\sim D}[c]$.

Proof. Rejection sampling produces a sample set S' drawn independently from $\hat{D}$. By assumption $A(S')$ outputs a classifier h such that

$$E_{x,y,c\sim\hat{D}}[I(h(x) \neq y)] \leq \epsilon$$

By the translation theorem (Proposition IV.A.1), we know that

$$E_{x,y,c\sim\hat{D}}[I(h(x) \neq y)] = \frac{1}{N} E_{x,y,c\sim D}[c\, I(h(x) \neq y)]$$

Thus,

$$\frac{1}{N} E_{x,y,c\sim D}[c\, I(h(x) \neq y)] \leq \epsilon$$

and

$$E_{x,y,c\sim D}[c\, I(h(x) \neq y)] \leq \epsilon N$$

□

IV.C.3 Sample complexity of cost-proportionate rejection sampling

The accuracy of a learned classifier generally improves monotonically with the number of samples in the training set. Since cost-proportionate rejection sampling produces a smaller training set (by a factor of about N/Z), worse performance than observed using the original dataset may result.

This turns out to not be the case, in the *agnostic PAC-learning model* [85, 42, 47], which formalizes the notion of probably approximately optimal learning from arbitrary distributions D.

Definition IV.C.1. *A learning algorithm A is said to be an* agnostic PAC-learner *for hypothesis class H, with sample complexity $m(1/\epsilon, 1/\delta)$ if for all $\epsilon > 0$ and $\delta > 0$, $m = m(1/\epsilon, 1/\delta)$ is the least sample size such that for all distributions D (over $X \times Y$), the classification error rate of its output h is at most ϵ more than the best achievable by any member of H with probability at least $1 - \delta$, whenever the sample size exceeds m.*

By analogy, we can formalize the notion of *cost-sensitive* agnostic PAC-learning.

Definition IV.C.2. *A learning algorithm A is said to be a* cost-sensitive agnostic PAC-learner *for hypothesis class H, with* cost-sensitive *sample complexity $m(1/\epsilon, 1/\delta)$, if for all $\epsilon > 0$ and $\delta > 0$, $m = m(1/\epsilon, 1/\delta)$ is the least sample size such that for all distributions D (over $X \times Y \times C$), the expected cost of its output h is at most ϵ more than the best achievable by any member of H with probability at least $1 - \delta$, whenever the sample size exceeds m.*

We will now use this formalization to compare the cost-sensitive PAC-learning sample complexity of two methods: applying a given base classifier learn-

ing algorithm to a sample obtained through cost-proportionate rejection sampling, and applying the same algorithm on the original training set. We show that the cost-sensitive sample complexity of the latter method is lower-bounded by that of the former.

Theorem IV.C.2. *(Sample Complexity Comparison) Fix an arbitrary base classifier learning algorithm A, and suppose that $m_{orig}(1/\epsilon, 1/\delta)$ and $m_{rej}(1/\epsilon, 1/\delta)$, respectively, are cost-sensitive sample complexity of applying A on the original training set, and that of applying A with cost-proportionate rejection sampling. Then, the following holds.*

$$m_{orig}(1/\epsilon, 1/\delta) = \Omega(m_{rej}(1/\epsilon, 1/\delta)).$$

Proof. Let $m(1/\epsilon, 1/\delta)$ be the (cost-insensitive) sample complexity of the base classifier learning algorithm A. (If no such function exists, then neither $m_{\text{orig}}(1/\epsilon, 1/\delta)$ nor $m_{\text{rej}}(1/\epsilon, 1/\delta)$ exists, and the theorem holds vacuously.) Since Z is an upper bound on the cost of misclassifying an example, we have that the cost-sensitive sample complexity of using the original training set satisfies

$$m_{\text{orig}}(1/\epsilon, 1/\delta) = \Theta(m(Z/\epsilon, 1/\delta))$$

This is because given a distribution (over $X \times Y$) that forces ϵ more classification error than optimal, another distribution (over $X \times Y \times C$) can be constructed, that forces ϵZ more cost than optimal, by assigning cost Z to all examples on which A errs.

Now from Theorem IV.C.1 and noting that the central limit theorem implies that cost-proportionate rejection sampling reduces the sample size by a factor of $\Theta(N/Z)$, the cost-sensitive sample complexity for cost-proportionate rejection sampling is:

$$m_{\text{rej}}\left(1/\epsilon, 1/\delta\right) = \Theta\left(\frac{Z}{N} m\left(N/\epsilon, 1/\delta\right)\right). \tag{IV.C.1}$$

A fundamental theorem from PAC-learning theory [25] states that

$$m(1/\epsilon, 1/\delta) = \Omega((1/\epsilon)\ln(1/\delta)).$$

When

$$m(1/\epsilon, 1/\delta) = \Theta((1/\epsilon)\ln(1/\delta)),$$

Equation (IV.C.1) implies:

$$\begin{aligned} m_{\text{rej}}(1/\epsilon, 1/\delta) &= \Theta\left(\frac{Z}{N}\frac{N}{\epsilon}\ln(1/\delta)\right) \\ &= \Theta\left(\frac{Z}{\epsilon}\ln(1/\delta)\right) \\ &= \Theta\left(m_{\text{orig}}(1/\epsilon, 1/\delta)\right) \end{aligned}$$

Finally, note that when $m(1/\epsilon, 1/\delta)$ grows faster than linear in $1/\epsilon$, we have

$$m_{\text{rej}}(1/\epsilon, 1/\delta) = o(m_{\text{orig}}(1/\epsilon, 1/\delta)),$$

which finishes the proof. □

Note that the linear dependence of sample size on $1/\epsilon$ is only achievable by an ideal learning algorithm, and in practice super-linear dependence is expected, especially in the presence of noise. Thus, the above theorem implies that cost-proportionate rejection sampling minimizes cost better than no sampling for worst case distributions.

This is a remarkable property about any sampling scheme, since one generally expects that predictive performance is compromised by using a smaller sample. Cost-proportionate rejection sampling seems to *distill* the original sample and obtains a sample of smaller size, which is at least as informative as the original.

IV.C.4 Costing

From the same original training sample, different runs of rejection sampling will produce different training samples. Furthermore, the fact that rejection sampling produces very small samples means that the computational time required for learning a classifier is generally much smaller.

We can take advantage of these properties to devise an ensemble learning algorithm based on repeatedly performing rejection sampling from S to produce

Costing(Learner A, Sample Set S, count t)

1. **For** $i = 1$ **to** t **do**
   (a) $S' =$ **rejection sample from** S
   (b) **Let** $h_i \equiv A(S')$
2. **Output** $h(x) = \textbf{sign}\left(\sum_{i=1}^{t} h_i(x)\right)$

Table IV.1: The costing algorithm.

multiple sample sets $S'_1, ..., S'_m$, and then learning a classifier for each set. The output classifier is the average over all learned classifiers. We call this technique *costing*. The pseudo-code for costing is shown in table IV.1.

The goal in averaging is to improve performance. There are several empirical and theoretical pieces of evidence suggesting that averaging can be useful. On the empirical side, many people have observed good performance from bagging despite throwing away a $1/e$ fraction of the samples (and the weakened overfitting control as noted earlier). On the theoretical side, there has been considerable work which proves that the "complexity" (ability to overfit) of an average of classifiers might be smaller than naively expected when a large margin exists. The preponderance of learning algorithms producing averaging classifiers provides significant evidence that averaging is useful.

Note that despite the extra computational cost of averaging, the overall computational time of costing is generally much smaller than for a learning algorithm using sample set S (with or without weights). This is the case because most learning algorithms have running times that are super-linear in the number of examples.

KDD-98:

Method	Without Weights	With Weights
Naive Bayes	0.24	12367
Boosted NB	-1.36	14489
C4.5	0	118
SVMLight	0	13683

DMEF-2:

Method	Without Weights	With Weights
Naive Bayes	16462	32608
Boosted NB	121	36381
C4.5	0	478
SVMLight	0	36443

Table IV.2: Test set profits on the KDD-98 and DMEF-2 datasets using the transparent box approach.

IV.D Experimental results

We show empirical results using the two real-world cost-sensitive datasets described in Chapter II, Section II.C: KDD-98 and DMEF-2.

IV.D.1 Transparent box results

Table IV.2 shows results obtained when we apply Naive Bayes, boosted Naive Bayes (100 iterations) C4.5 and SVMLight to both the KDD-98 and the DMEF-2 datasets, with and without the importance weights.

Without giving the costs as weights, the classifiers label very few of the examples positive, resulting in small (and even negative) profits. With the costs given as weights to the learners, the results improve significantly for all learners, except C4.5. Cost-sensitive boosted Naive Bayes gives results comparable to the results obtained in Chapter III.

We optimized the parameters of the SVM by cross-validation on the training set. Without weights, no setting of the parameters prevented the algorithm of labeling all examples as negatives. With weights, the best parameters were a polynomial kernel with degree 3 and $C = 5 \times 10^{-5}$ for KDD-98 and a linear kernel

with $C = 0.0005$ for DMEF-2. However, even with this parameter setting, the results are not so impressive. This may be a relatively hard problem for margin-based classifiers because the data is very noisy. Note also that running SVMLight on this dataset takes about 3 orders of magnitude longer than AdaBoost with 100 iterations on the same machine.

The failure of C4.5 to achieve good profits when given the costs as weights is probably related to the fact that the standard facility for incorporating weights provided in the algorithm is heuristic. So far, it has been used only in situations where the weights are fairly uniform (such as is the case for fractional instances due to missing data). These results indicate that it might not be suitable for situations with highly non-uniform costs. The fact that it is non-trivial to incorporate costs directly into existing learning algorithms is the motivation for the black box approaches that we present here.

IV.D.2 Black box results

Table IV.3 shows the results of applying the same learning algorithms to the KDD-98 and DMEF-2 data using resampled training sets of different sizes. For each size, we repeat the experiments 10 times with different resampled sets to get mean and standard error (in parentheses). The training set profits are on the original training set from which we draw the resampled sets.

These results here confirm that straightforward application of resampling to implement the black box approach can result in very poor performance, as we remarked earlier. The poor performance of resampling is essentially due to overfitting. When there are large differences in the magnitude of importance weights, it is typical for an example to be picked twice (or more). In table IV.3, we see that as we increase the resampled training set size and, as a consequence, the number of duplicate examples in the training set, the more overfitting we see for C4.5. The training profit becomes larger while the test profit becomes smaller.

Examples which appear multiple times in the training set of a learning

1000:

	KDD-98		DMEF-2	
	Training	Test	Training	Test
NB	11251 (330)	10850 (325)	33298 (495)	34264 (419)
BNB	11658 (311)	11276 (383)	33902 (558)	30304 (660)
C4.5	11124 (255)	9548 (331)	37905 (1467)	24011 (1931)
SVM	10320 (372)	10131 (281)	28837 (1029)	30177 (1196)

10000:

	KDD-98		DMEF-2	
	Training	Test	Training	Test
NB	12811 (155)	11993 (185)	32742 (793)	33956 (798)
BNB	13838 (65)	12886 (212)	34802 (806)	31342 (772)
C4.5	22083 (271)	7599 (310)	67960 (763)	9188 (458)
SVM	11228 (182)	11015 (161)	31263 (1121)	32585 (891)

100000:

	KDD-98		DMEF-2	
	Training	Test	Training	Test
NB	12531 (242)	12026 (256)	33511 (475)	34506 (405)
BNB	14107 (152)	13135 (159)	34505 (822)	31889 (733)
C4.5	40704 (152)	2259 (107)	72574 (1205)	3149 (519)
SVM	13565 (129)	12808 (220)	34309 (719)	33674 (600)

Table IV.3: Profits on the KDD-98 and DMEF-2 datasets using resampling.

algorithm can defeat complexity control mechanisms built into learning algorithms. For example, suppose that we have a decision tree algorithm which divides the training data into a "growing set" (used to construct a tree) and a "pruning set" (used to prune the tree for complexity control purposes). If the pruning set contains examples which appear in the growing set, the complexity control mechanism is defeated.

This observation has implications for the practice of bagging, although the loss of complexity control is not as severe as observed here. Uniform resampling from a set of size m, m times produces only about $\frac{m}{e}$ duplicates. Since (with high probability) not all examples are duplicates, complexity control in bagging is only weakened and not removed. We note that Fan et al [32] have proposed a modification to bagging, in which partitioning is used in place of resampling to address this issue.

Although not as markedly as for C4.5, we see the same phenomenon for the other learning algorithms. In general, as the size of the resampled size grows, the larger is the difference between the training set profit and test set profit. And, even with 100000 examples, we do not obtain the same test set results as giving the weights directly to Boosted Naive Bayes and SVM.

The fundamental difficulty here is that the samples in S' are not drawn *independently* from $\hat{D}$. In particular, if $\hat{D}$ is a density, the probability of observing the same example twice given independent draws is 0, while the probability using resampling is greater than 0. Thus resampling-with-replacement fails because the resampled set S' is not constructed independently.

Figures IV.D.1 and IV.D.2 shows the results of costing on the KDD-98 and DMEF-2 datasets, with the base learners and $Z = 200$ or $Z = 6247$, respectively. We repeated the experiment 10 times for each t and calculated the mean and standard error of the profit. The results for $t = 1$, $t = 100$ and $t = 200$ are also given in table IV.4.

In the KDD-98 case, each resampled set has only about 600 examples,

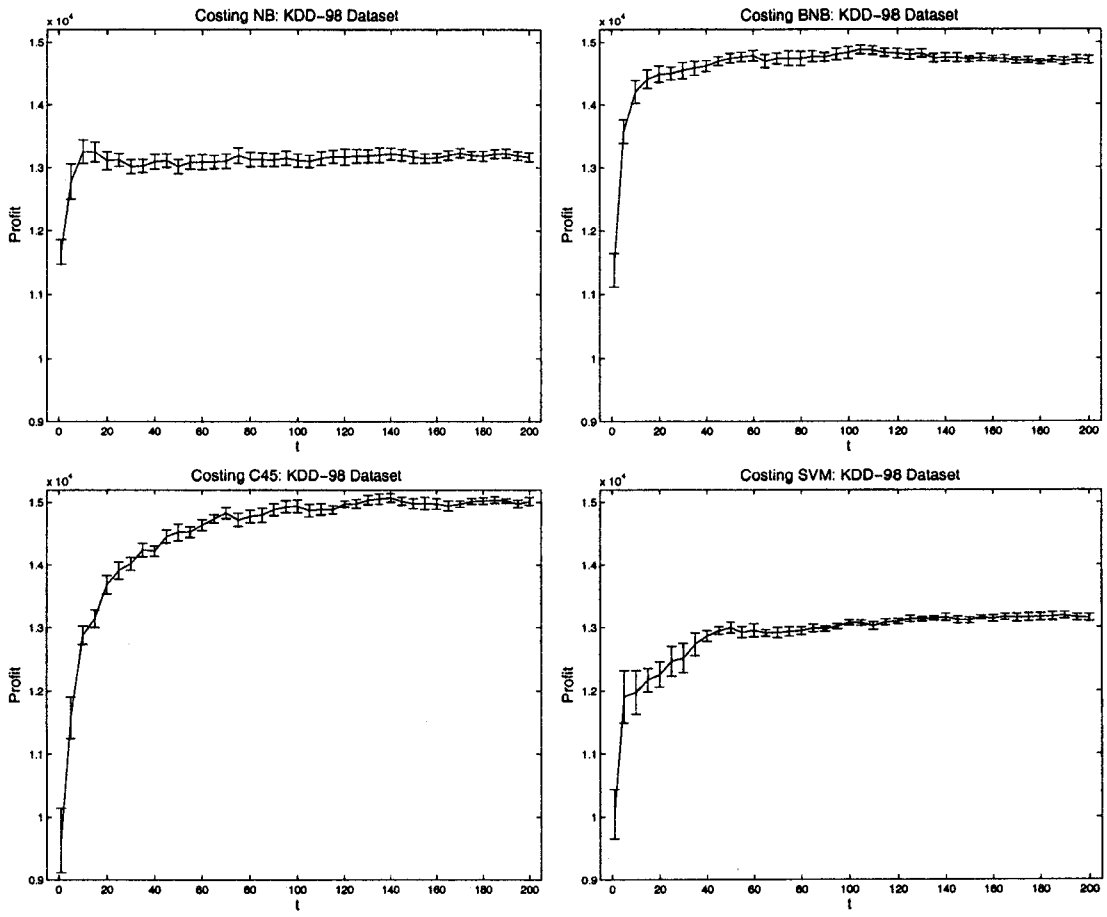


Figure IV.D.1: Costing. The graphs shows how the KDD-98 test set profit grows as the number of resampled sets is increased from 1 to 200.

KDD-98:

	1	100	200
NB	11667 (192)	13111 (102)	13163 (68)
BNB	11377 (263)	14829 (92)	14714 (62)
C4.5	9628 (511)	14935 (102)	15016 (61)
SVM	10041 (393)	13075 (41)	13152 (56)

DMEF-2:

	1	100	200
NB	26287 (3444)	37627 (335)	37629 (139)
BNB	24402 (2839)	37376 (393)	37891 (364)
C4.5	27089 (3425)	36992 (374)	37500 (307)
SVM	21712 (3487)	33584 (1215)	35290 (849)

Table IV.4: Test set profits on the KDD-98 and DMEF-2 datasets using costing.

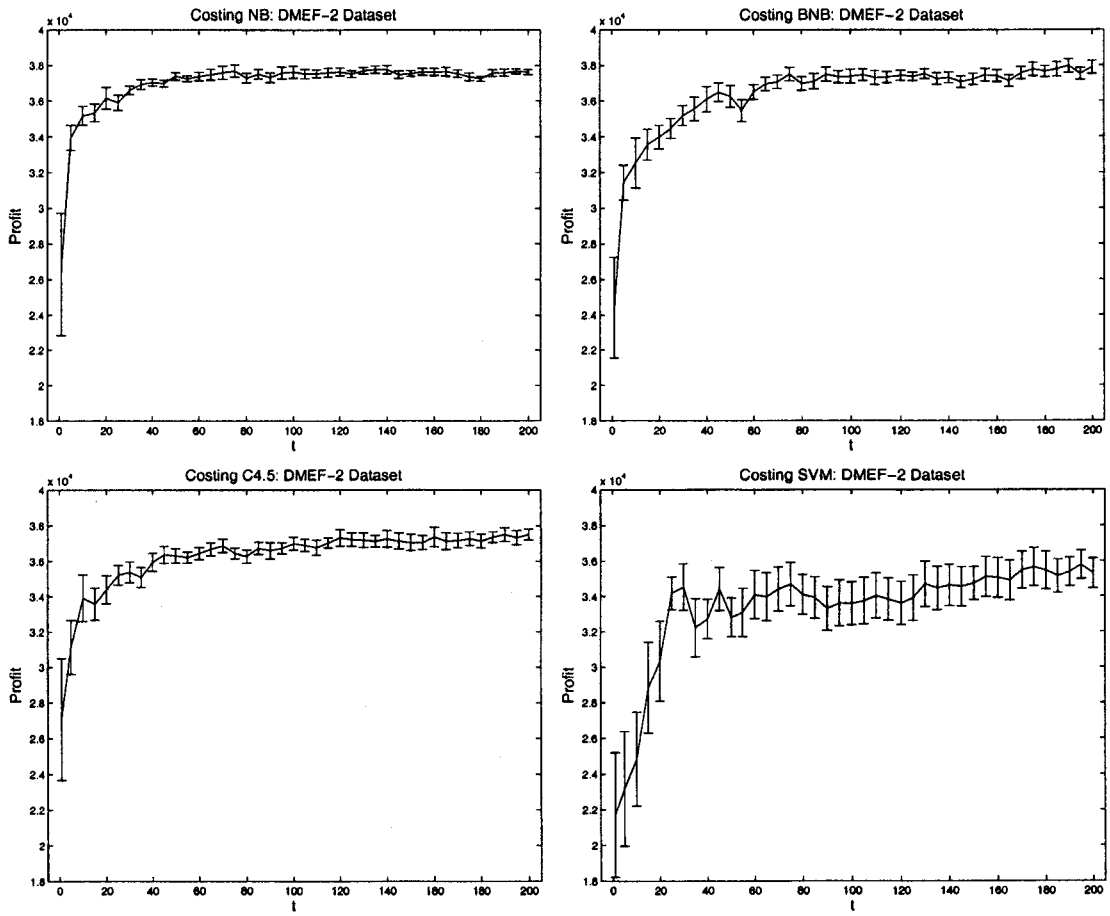


Figure IV.D.2: Costing. The graphs shows how the DMEF-2 test set profit grows as the number of resampled sets is increased from 1 to 200.

because the importance of the examples varies from 0.68 to 199.32 and there are few "important" examples. About 55% of the examples in each set are positive, even though on the original dataset the percentage of positives is only 5%.

With $t = 200$, the C4.5 version yields profits around \$15000, which is exceptional performance for this dataset. In particular, these results are better than the ones obtained with C4.5 using direct cost-sensitive decision-making (Chapter III, table III.3), which yielded profits around 14000. The results with Naive Bayes (profits around 13000), however, are worse than direct cost-sensitive decision-making which yields profits around 15000.

In the DMEF-2 case, each set has only about 35 examples, because the importances vary even more widely (from 2 to 6246) and there are even fewer examples with a large importance than in the KDD-98 case. The percentage of positive examples in each set is about 50%, even though on the original dataset it was only 2.5%.

For learning the SVMs, we used the same kernels as we did in section IV.B and the default setting for C. In that section, we saw that by feeding the weights directly to the SVM, we obtain a profit of \$13683 on the KDD-98 dataset and of \$36443 on the DMEF-2 dataset. Here, we obtain profits around \$13100 and \$35000, respectively. However, this did not require parameter optimization and, even with $t = 200$, was much faster to train. The reason for the speed-up is that the time complexity of SVM learning is generally super-linear in the number of training examples.

IV.E Conclusions

Costing is a technique which produces a cost-sensitive classifier using only black box access to a classifier learning method. Conceptually, it is much simpler than the methods for cost-sensitive learning by expected cost estimation presented in Chapter III, because it does not require accurate class membership probability

estimates from the classifier and avoids the estimation of costs.

Furthermore, it is fast, results in good performance for a variety of classifier learners and often achieves drastic savings in computational resources, particularly with respect to space requirements. This last property is especially desirable in applications of cost-sensitive learning to domains that involve massive amounts of data, such as fraud detection, targeted marketing, and intrusion detection.

Another desirable property of any reduction is that it applies to the theory as well as to concrete algorithms. Thus, the reduction presented in this chapter allows us to automatically apply any future results in classifier learning to cost-sensitive learning. For example, a bound on the future error rate of $A(S')$ implies a bound on the expected cost with respect to the distribution D. This additional property of a reduction is especially important because cost-sensitive learning theory is still young and relatively unexplored.

A disadvantage of costing is that, since it uses the importance formulation, it only applies to two-class problems. Thus, a direction for future research is the design of a similar algorithm for multiclass problems. If there are K classes, the minimal representation of costs is $K - 1$ weights for each example, because we have to account for the importance of misclassifying the example into each of the incorrect classes. Margineantu [57] presents and evaluates different heuristic methods for dealing with these weights in the case of fixed costs, such as using the maximum weight or taking the average. Nonetheless, a general reduction from example-dependent cost-sensitive learning to cost-insensitive learning is still an open problem.

Acknowledgments

The text of this chapter, in part, is a reprint of the material as it will appear in the *Proceedings of the 2003 IEEE International Conference on Data Mining* [97]. The dissertation author was the primary author, and the co-authors listed in this publication directed and supervised the research which forms the basis for this chapter.

Chapter V

Calibrating classifier scores

Most existing learning methods produce classifiers that output ranking scores along with the class label, but they do not output accurate class probability estimates. In this chapter, we present a new method for obtaining calibrated two-class probability estimates that can be applied to any classifier that produces a ranking of examples. Besides being fast and very simple to understand and implement, our method produces probability estimates that are comparable to or better than the ones produced by other methods.

Current methods for transforming ranking scores into probability estimates apply only to two-class problems. In this chapter, we present a method for obtaining accurate multiclass probability estimates from ranking scores: we decompose the multiclass problem into a series of binary problems, learn a classifier for each one of them, calibrate the scores from each classifier, and combine them to obtain multiclass probabilities.

In Section V.A, we motivate the need for classifiers to output estimates of class membership probabilities. In Section V.B, we review the notion of calibration of probability estimates and show that although the scores produced by naive Bayes and support vector machine (SVM) classifiers tend to rank examples well, they are not well-calibrated. In Section V.C, we review previous methods for mapping two-class scores into probability estimates, explain their shortcomings and present

our new method. In Section V.D we discuss how to combine calibrated two-class probability estimates into calibrated multiclass probability estimates. In Section V.E we present an experimental evaluation of these methods applied to naive Bayes and SVM scores in a variety of domains. Finally, in Section V.F we summarize the contributions in this chapter and suggest directions for future work in calibration.

V.A The need for class membership probability estimates

Most classifier learning methods produce classifiers that output scores $s(x)$ for an example x, which can be used to rank examples from the most probable member to the least probable member of a class c. That is, for two examples x and y, if $s(x) < s(y)$ than $P(c|x) < P(c|y)$.

However, in many applications, a ranking of examples according to class membership probability is not enough. What is needed is an accurate estimate of the probability that each example is a member of the class of interest.

Class membership probability estimates are important when the classification outputs are not used in isolation but are combined with other sources of information for decision-making. For example, in Chapter III, we presented methods for cost-sensitive learning that depended on combining class membership probability estimates with misclassification costs for accomplishing cost-sensitive classification.

Another example is the use of classification as a component to a high-level system that combines classifier results. This is the case, for example, in handwritten character recognition and in speech recognition, where the outputs from a classifier that recognizes single characters or phonemes are combined using a Viterbi search or HMM [11].

Also, active learning based on uncertainty sampling [52] requires class membership probability estimates. In this framework, the learner asks a teacher to label a set of examples that is sampled from a pool of unlabeled examples with

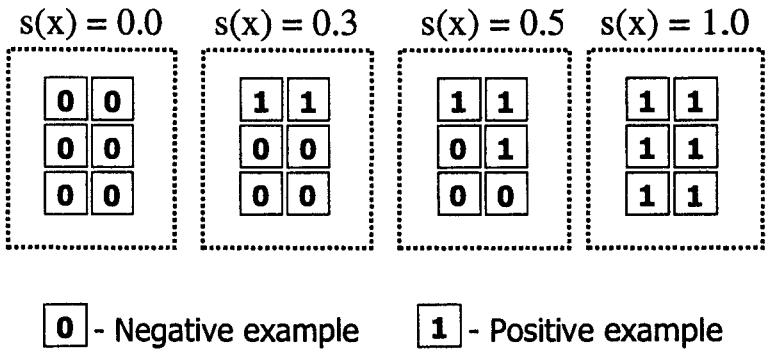


Figure V.B.1: The concept of calibration.

weights that are inversely proportional to the class membership probabilities.

V.B Calibration definition and examples

Assume that we have a classifier that for each example x outputs a score $s(x)$ between 0 and 1. This classifier is said to be well-calibrated if the empirical class membership probability $P(c|s(x) = s)$ converges to the score value $s(x) = s$, as the number of examples classified goes to infinity [59]. Intuitively, if we consider all the examples to which a classifier assigns a score $s(x) = 0.8$, then 80% of these examples should be members of the class in question. Figure V.B.1 illustrates this concept. Calibration is important if we want the scores to be directly interpretable as the chances of membership in the class.

The calibration of a classifier can be visualized through a reliability diagram [18]. In the case where there is a small number of possible score values, for each score value s, we compute the empirical probability $P(c|s(x) = s)$: the number of examples with score s that belong to class c divided by the total number of examples with score s. We then plot s versus $P(c|s(x) = s)$. If the classifier is well-calibrated, all points fall into the $x = y$ line, indicating that the scores are equal to the empirical probability.

However, in practical situations, the number of possible scores is large

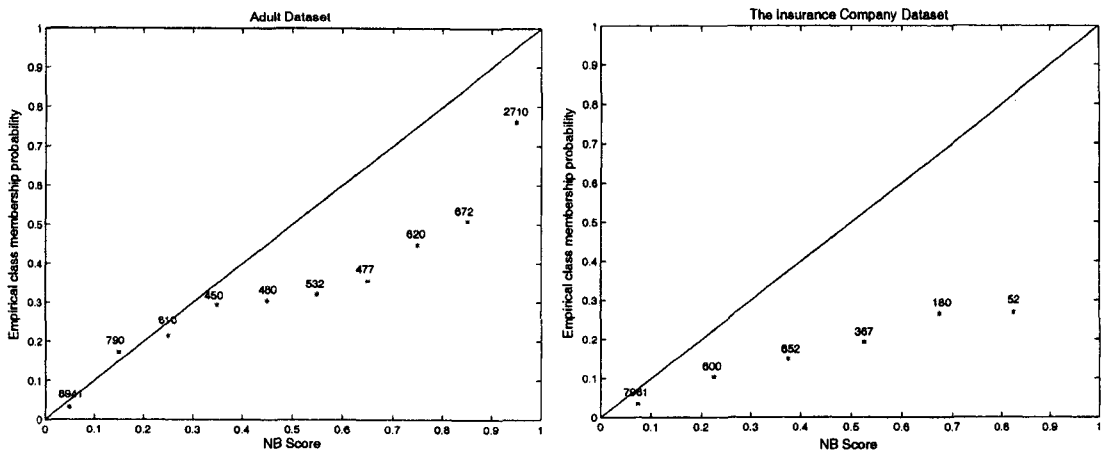


Figure V.B.2: Reliability diagrams for NB. The numbers indicate how many examples fall into each bin (test set).

compared to the number of available test examples, so we cannot calculate reliable empirical probabilities for each possible score value. In this case, we can resort to discretizing the score space. But because the scores are not uniformly distributed, we have to carefully choose bin sizes so that there are enough examples to calculate reliable empirical probability estimates for each bin.

V.B.1 Naive Bayes

Naive Bayesian classifiers assign to each test example a score between 0 and 1 that can be interpreted, in principle, as a class membership probability estimate. However, it is well known that these scores are not well-calibrated [21].

Naive Bayes is based on the assumption that the attributes of examples are independent given the class of the examples. Because attributes tend to be correlated in real data, the scores $s(x)$ produced by naive Bayes are typically too extreme: for most x, either $s(x)$ is near 0 and then $s(x) < P(c|x)$ or $s(x)$ is near 1 and then $s(x) > P(c|x)$. However, naive Bayesian classifiers tend to rank examples well: if $s(x) < s(y)$ then $P(c|x) < P(c|y)$.

In Figure V.B.2 we show reliability diagrams for two well-known datasets:

Adult and TIC (see Section V.E for information on these datasets), where the score space has been discretized into bins of size 0.1 and 0.15, respectively. As we can see in the graphs, although tending to vary monotonically with the empirical probability, naive Bayes scores are not well-calibrated because many of the points do not fall into the $x = y$ line.

V.B.2 Support Vector Machines

For each test example x, an SVM classifier outputs a score that is the distance of x to the hyperplane learned for separating positive examples from negative examples. The sign of the score indicates if the example is classified as positive or negative. The magnitude of the score can be taken as a measure of confidence in the prediction, since examples far from the separating hyperplane are presumably more likely to be classified correctly.

Although the range of SVM scores is $[-a, a]$ (where a depends on the problem), we can map the scores into the $[0, 1]$ interval by re-scaling them. If $f(x)$ is the original score, then

$$s(x) = (f(x) + a)/2a \tag{V.B.1}$$

is a re-scaled score between 0 and 1, such that if $f(x) > 0$ then $s(x) > 0.5$ and if $f(x) < 0$ then $s(x) < 0.5$. However, these scores tend to not be well-calibrated since the distance from the separating hyperplane is not exactly proportional to the chances of membership in the class.

In Figure V.B.3 we show reliability diagrams for re-scaled SVM scores using the Adult and TIC datasets, where the score spaces are discretized into bins of size 0.08 and 0.15, respectively. We see that SVM scores vary monotonically with the empirical probability, but are not well-calibrated.

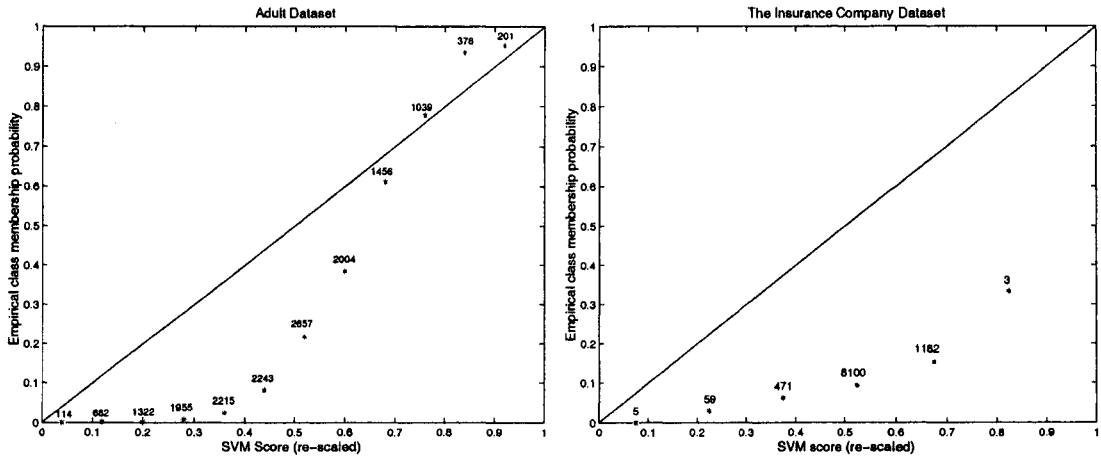


Figure V.B.3: Reliability diagrams for SVM. The numbers indicate how many examples fall into each bin (test set).

V.C Mapping scores into probability estimates

Suppose we have a set of examples for which we know the true labels. In this case we can assume that $P(c|x) = 1$ for positive examples and $P(c|x) = 0$ for negative examples. If we apply the classifier to those examples to obtain scores $s(x)$, we can learn a function mapping scores $s(x)$ into probability estimates $\hat{P}(c|x)$. If the learning method does not overfit the training data, we can use the same data to learn this function. Otherwise, we need to break the training data into two sets: one for learning the classifier and the other for learning the mapping function.

In any case, we need a regularization criterion to avoid learning a mapping function that does not generalize well to new data. One possible regularization criteria is to impose a particular parametric shape for the function and use the available data to learn parameters such that the function fits well the data according to some measure.

The parametric approach proposed by Platt [63] for SVM scores consists in finding the parameters A and B for a sigmoid function of the form

$$\hat{P}(c|x) = \frac{1}{1 + e^{As(x)+B}}$$

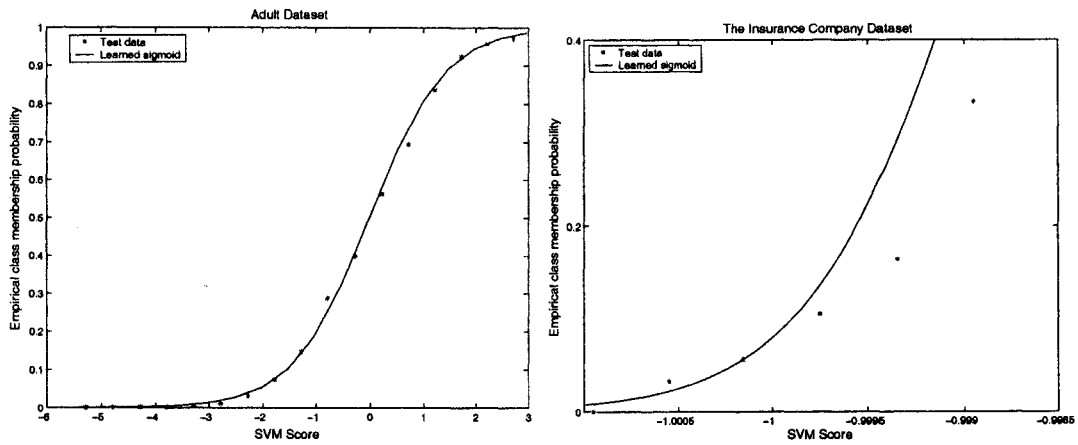


Figure V.C.1: Mapping SVM scores into probability estimates using a sigmoid function.

mapping the scores $s(x)$ into probability estimates $\hat{P}(c|x)$, such that the negative log-likelihood of the data is minimized.

This method is motivated by the fact that the relationship between SVM scores and the empirical probabilities $P(c|x)$ appears to be sigmoidal for many datasets. This is the case for the Adult dataset, as can be seen in Figure V.C.1, where we show the learned sigmoid using the training data and the empirical probabilities for the test data, for Adult and TIC. Platt has shown empirically that this method yields probability estimates that are at least as accurate as ones obtained by training an SVM specifically for producing accurate class membership probability estimates, while being faster.

The same method can be applied to naive Bayes. This was proposed by Bennett [8] for the Reuters dataset. In Figure V.C.2 we show the sigmoidal fit to the naive Bayes scores for the Adult and TIC datasets. The sigmoidal shape does not appear to fit naive Bayes scores as well as it fits SVM scores, for these datasets.

If the shape of the mapping function is unknown, we can resort to a non-parametric method such as binning [94]. In binning, the training examples are sorted according to their scores and the sorted set is divided into b subsets of

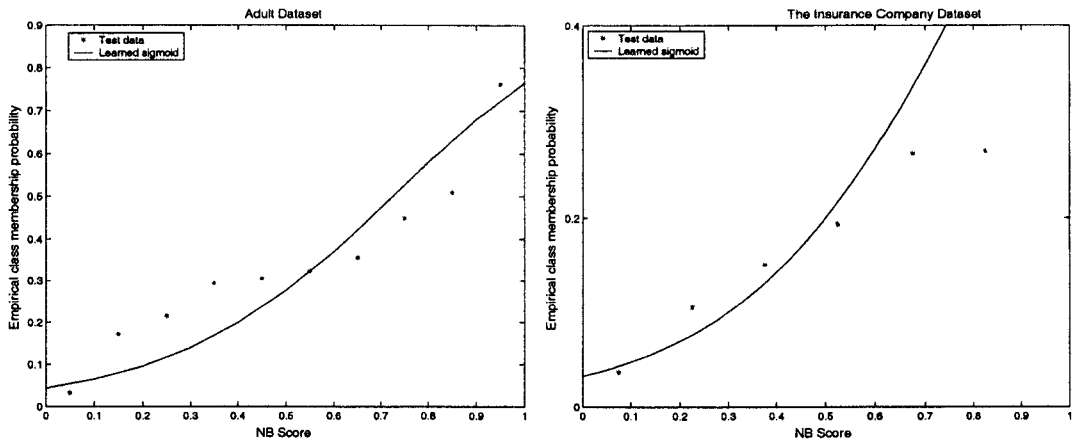


Figure V.C.2: Mapping NB scores into probability estimates using a sigmoid function.

equal size, called bins. For each bin we compute lower and upper boundary $s(\cdot)$ scores. Given a test example x, we place it in a bin according to its score $s(x)$. We then estimate the corrected probability that x belongs to class c as the fraction of training examples in the bin that actually belong to c.

A difficulty of the binning method is that we have to choose the number of bins by cross-validation. If the dataset is small, or highly unbalanced, cross-validation is not likely to indicate the optimal number of bins. Also, the size of the bins is fixed and the position of the boundaries is chosen arbitrarily. If the boundaries are such that we average together the labels of examples that clearly should have different probability estimates, the binning method will fail to produce accurate probability estimates.

We propose here an intermediary approach between sigmoid fitting and binning: isotonic regression [70]. Isotonic regression is a non-parametric form of regression in which we assume that the function is chosen from the class of all isotonic (i.e. non-decreasing) functions.

If we assume that the classifier ranks examples correctly, the mapping from scores into probabilities is non-decreasing, and we can use isotonic regression

to learn this mapping. A commonly used algorithm for computing the isotonic regression is pair-adjacent violators (PAV) [3]. This algorithm finds the stepwise-constant isotonic function that best fits the data according to a mean-squared error criterion.

PAV works as follows. Let $\{x_i\}_{i=1}^{N}$ be the training examples, $g(x_i)$ be the value of the function to be learned for each training example x_i, and g^* be the isotonic regression. If g is already isotonic, then we return $g^* = g$. Otherwise, there must be a subscript i such that $g(x_{i-1}) \leq g(x_i)$. The examples x_{i-1} and x_i are called pair-adjacent violators, because they violate the isotonic assumption. The values of $g(x_{i-1})$ and $g(x_i)$ are then replaced by their average, so that the examples x_{i-1} and x_i now comply with the isotonic assumption. If this new set of $n-1$ values is isotonic, then $g^*(x_{i-1}) = g^*(x_i) = (g(x_{i-1}+g(x_i))/2$, and $g^*(x_j) = g(x_j)$ otherwise. This process is repeated using the new values until an isotonic set of values is obtained. The computational complexity of this algorithm when implemented efficiently is $O(n)$. An efficient implementation of PAV in MATLAB is made available by Lutz Dümbgen [24].

When we apply this algorithm to the problem of mapping scores into probability estimates, we first sort the examples according to their scores and let $g(x_i)$ be 0 if x_i is negative, and 1 if x_i is positive. If the scores rank the examples perfectly, then all negative x_i come before the positive x_i and the values of g are not changed. The new probability estimate g^* is 0 for all negative examples and 1 for all positive examples. On the other hand, if the scores do not give any information about the ordering of the examples, g^* will be a constant function whose value is the average of all values of $g(x_i)$, which is the base rate of positive examples. Figure V.C.3 gives an example of the application of PAV.

In the general case, PAV will average out more examples in parts of the score space where the classifier ranks examples incorrectly, and less examples in parts of the space where the classifier ranks them correctly. We can view PAV as a binning algorithm where the position of the boundaries and the size of the bins

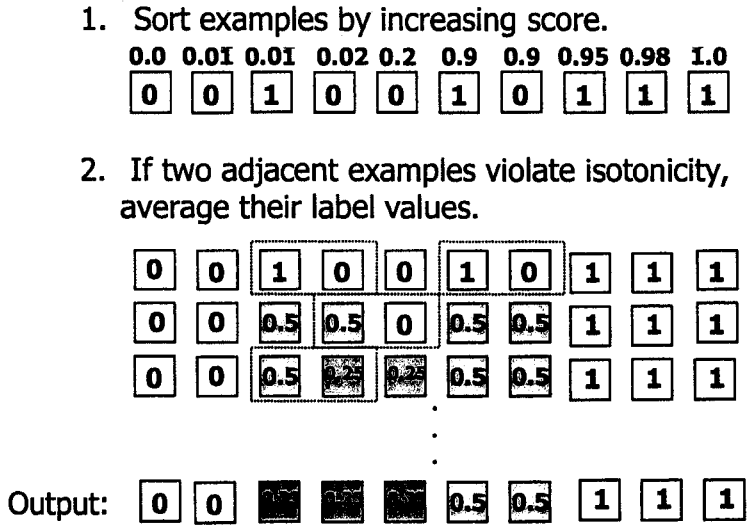


Figure V.C.3: The PAV algorithm in action.

are chosen according to how well the classifier ranks the examples.

PAV returns a set of intervals and an estimate $g(i)$ for each interval i, such that $g^*(i+1) \geq g^*(i)$. To obtain an estimate for a test example x, we find the interval i for which $s(x)$ is between the lowest and highest scores in the interval and assign $g^*(i)$ as the probability estimate for x.

In Figure V.C.4, we show the result of applying PAV to the Adult dataset, for both naive Bayes and SVM. The line shows the function that was learned on the training data, while the stars show empirical probabilities for the test data.

V.D Multiclass probability estimates

The notion of calibration introduced in Section V.B can be readily applied to multiclass probability estimates. Suppose we have a multiclass classifier that output scores $s(c_i|x)$ for each class c_i and each example x. The classifier is well-calibrated if, for each class c_i, the empirical probability $P(c_i|s(c_i|x) = s)$ converges to the score value $s(c_i|x) = s$, as the number of examples classified goes to infinity.

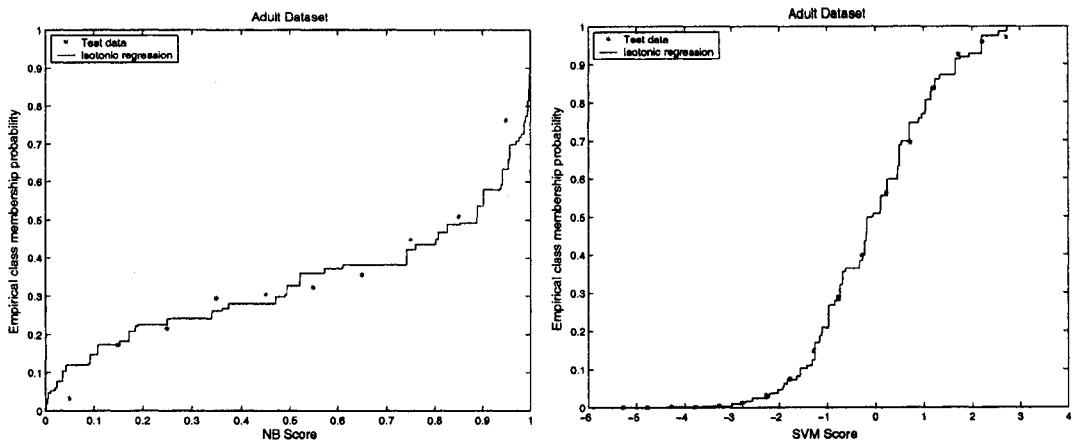


Figure V.C.4: Using the PAV algorithm to map naive Bayes and SVM scores into probability estimates.

However, the calibration methods discussed in Section V.C were designed exclusively for two-class problems. Mapping scores into probability estimates works well in the two-class case because we are mapping between one-dimensional spaces. In this setting, it is easy to impose sensible restrictions on the shape of the function being learned, as it is done with the sigmoidal shape or the monotonicity requirements.

In the general multiclass case, the mapping would have to be from $(k-1)$-dimensional space to another $(k-1)$-dimensional space. In this case, it is not clear which function shape should be imposed to the mapping function. Furthermore, because of the curse of dimensionality, non-parametric methods are not likely to yield accurate probabilities when the number of classes grows. For these reasons, we do not attempt to directly calibrate multiclass probability estimates. Instead, we first reduce the multiclass problem into a number of binary classification problems. Then we learn a classifier for each binary problems, and calibrate the scores from each classifier. Finally, we combine the binary probability estimates to obtain multiclass probabilities.

Two well-known approaches for reducing a multiclass problem to a set of

binary problems are known as one-against-all and all-pairs. In one-against-all, we train a classifier for each class using as positives the examples that belong to that class, and as negatives all other examples. In all-pairs, we train a classifier for each possible pair of classes ignoring the examples that do not belong to the classes in question.

Allwein *et al.* [1] represent any possible decomposition of a multiclass problem into binary problems by using a code matrix $M \in \{-1, 0, +1\}^{k \times l}$, where k is the number of classes and l is the number of binary problems. If $M(c, b) = +1$ then the examples belonging to class c are considered to be positive examples for the binary classification problem b. Similarly, if $M(c, b) = -1$ the examples belonging to c are considered to be negative examples for b. Finally, if $M(c, b) = 0$ the examples belonging to c are not used in training a classifier for b.

For example, in the 3-class case, the one-against-all code matrix is

	b_1	b_2	b_3
c_1	+1	+1	−1
c_2	−1	−1	+1
c_3	−1	−1	−1

and the all-pairs code matrix is

	b_1	b_2	b_3
c_1	+1	+1	0
c_2	−1	0	+1
c_3	0	−1	−1

These code matrices are a generalization of the error-correcting output coding (ECOC) scheme [19]. The difference is that ECOC does not allow zeros in the code matrix, meaning that all examples are used in each binary classification problem.

For an arbitrary code matrix M, we have an estimate $r_b(x)$ for each column b of M, such that

$$r_b(x) = P(\bigvee_{c \in I} c | \bigvee_{c \in I \cup J} c,\ x) = \frac{\sum_{c \in I} P(c|x)}{\sum_{c \in I \cup J} P(c|x)}$$

where I and J are the set of classes for which $M(\cdot, b) = 1$ and $M(\cdot, b) = -1$, respectively. We would like to obtain a set of probabilities $P(c|x)$ for each example x compatible with the $r_b(x)$ and subject to $\sum_c P(c|x) = 1$. Because there are $k-1$ free parameters and l constraints, and we generally consider matrices for which $l > k-1$, this is an over-constrained problem for which there is no exact solution.

Two approaches have been proposed for finding an approximate solution for this problem. The first is a least-squares method with non-negativity constraints proposed by Kong and Dietterich [49]. They have proposed this method for the original ECOC matrices, but it can easily be applied to arbitrary matrices. They test it on binary probability estimates from decision trees classifiers learned using C4.5, which are known not to be well-calibrated [95, 64]. Using synthetic data, they show that this method produces better estimates than multiclass C4.5.

The alternative method is called coupling, an iterative algorithm that finds the best approximate solution minimizing log-loss instead of squared error proposed by Zadrozny[93]. This method is an extension to the pairwise coupling method by Hastie and Tibshirani[39], which only applies to all-pairs matrices. The pseudo-code for coupling is given in table V.1. The algorithm was tested using boosted naive Bayes [26] as the binary learner, whose scores tend to be even less calibrated than naive Bayes scores because they are more extreme.

It is an open question which of the two existing methods for combining binary probability estimates yields the most accurate multiclass probability estimates. A desirable property for such a method is that the better calibrated the binary estimates are, the better calibrated the multiclass estimates should be. In the next section, we compare these methods experimentally on two multiclass datasets.

1. Start with some guess for the $\hat{P}(c|x)$ and corresponding $\hat{r}_b(x)$.

2. Repeat until convergence:

   (a) For each c

$$\hat{P}(c|x) \leftarrow \hat{P}(c|x)\frac{\sum_{b \text{ s.t. } M(c,b)=1} n_b r_b(x) + \sum_{b \text{ s.t. } M(c,b)=-1} n_b(1 - r_b(x))}{\sum_{b \text{ s.t. } M(c,b)=1} n_b \hat{r}_b(x) + \sum_{b \text{ s.t. } M(c,b)=-1} n_b(1 - \hat{r}_b(x))}$$

   (b) Re-normalize the $\hat{P}(c|x)$.

   (c) Recompute the $\hat{r}_b(x)$.

Table V.1: The coupling algorithm.

V.E Experimental Evaluation

Here we present results of the application of the methods discussed in the previous sections to a variety of datasets. Since the methods used for learning the classifiers do not overfit the training data for these datasets, in all experiments we use the same data for learning both the classifier and the calibration functions.

As the primary metric for assessing the accuracy of probability estimates, we use the mean squared error (MSE), also known as the Brier score [16]. For one example x, the squared error (SE) is defined as

$$\text{SE}(x) = \sum_c (T(c|x) - P(c|x))^2$$

where $P(c|x)$ is the probability estimated for example x and class c and $T(c|x)$ is defined to be 1 if the actual label of x is c and 0 otherwise. We calculate the SE for each example in the training and test sets to obtain the MSE for each set.

DeGroot and Fienberg [18] show that the MSE can be separated into two components, one measuring calibration and the other measuring refinement. If the classifier is well-calibrated the first component is zero. For two classifiers that are well-calibrated, the one for which the probability estimates $P(c|x)$ are closer to 0 or 1 is said to be more refined, because it makes predictions that are more

	MSE		Profit	
Method	Training	Test	Training	Test
NB	0.10089	0.10111	$10083	$9531
Sigmoid NB	0.09542	0.09533	$14134	$14120
PAV NB	0.09522	0.09528	$15685	$14447

Table V.2: MSE and profit on the KDD-98 dataset.

confident. If the two classifiers are well-calibrated, the one with the lowest MSE is more refined, and thus, preferable.

Although MSE can be applied in general, it is more sensible to evaluate the quality of probability estimates in practical situations using a domain-specific metric. For example, in direct mailing, we should evaluate how good the probability estimates are by the profit obtained when we mail people according to a policy that uses the estimates. MSE tends to be correlated with profit [94], so when we do not have domain-specific information to calculate profit we can use MSE to evaluate our methods.

When the classifier is to be used for classification in a domain where errors are equally costly, we can evaluate the probability estimates by assigning each example x to its most likely class $c^*(x) = \text{argmax}_c P(c|x)$ and calculating the error rate. This metric does not assess the calibration of the estimates directly, but calibration can potentially improve the error rate because the most likely class can change.

V.E.1 Two-class problems

The first dataset we use is the KDD-98 dataset, which is described in detail in Chapter II, section II.C.

As we have seen in Chapter III, the optimal mailing policy for this domain is to solicit people for whom the expected return

$$P(\text{donation}|x)y(x)$$

is greater than the cost of mailing a solicitation, where $y(x)$ is the estimated

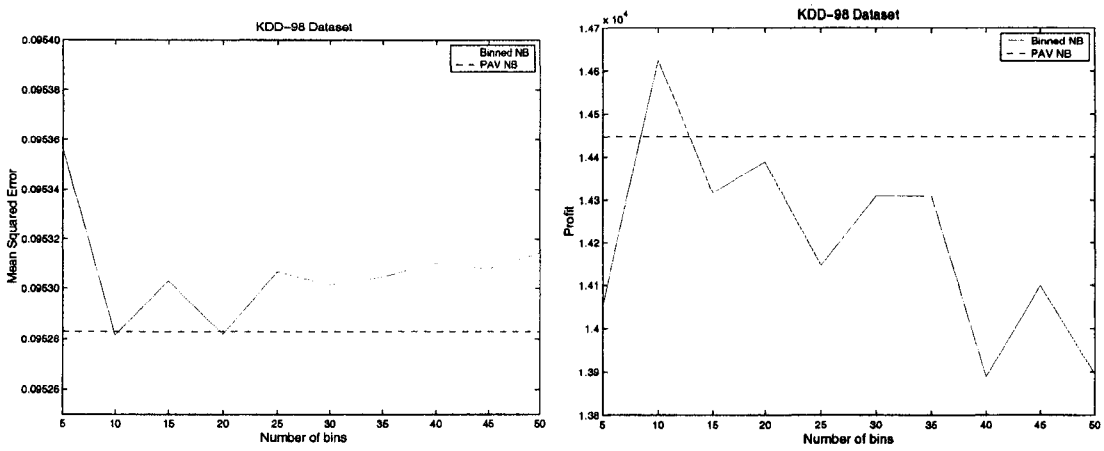


Figure V.E.1: Binning and PAV on the KDD-98 dataset. The figure on the left shows results with Naive Bayes and the figure on the right shows results with SVM.

donation amount. We use fixed values for $y(x)$ obtained using linear regression as done in Chapter III, section III.D).

We use naive Bayes to estimate $P(\text{donation}|x)$ and apply each of the calibration methods discussed in Section V.C. Table V.2 shows MSE and profits for the raw naive Bayes scores and the calibrated scores obtained using sigmoid fitting and PAV. As expected, MSE and profit are significantly improved by calibration. Although PAV overfits slightly the training data, it performs better than sigmoid fitting. In figure V.E.1, we compare PAV to binning with bin sizes varying from 5 to 50. Although we did not have to set any parameters for the PAV method, it performed comparably to the best parameter setting for binning, both for naive Bayes and for SVM.

The next dataset we use is The Insurance Company Benchmark (TIC), also known as the COIL 2000 dataset, which is available in the UCI KDD repository [6]. The decision-making task is analogous to the KDD-98 task: deciding which customers to offer a caravan insurance policy. This dataset is also divided in a standard way into a training set (5822 examples) and a test set (4000 examples). We use the same attributes as used for the winning entry of the COIL 2000

Method	Training	Test
NB	0.12845	0.13551
Sigmoid NB	0.10536	0.10905
PAV naive Bayes	0.10315	0.10818
SVM	0.11942	0.11889
Sigmoid SVM	0.11080	0.11122
PAV SVM	0.10974	0.11200

Table V.3: MSE on the TIC dataset.

challenge [29].

Using the training set, we learn a model for the probability that a customer has acquired a caravan insurance policy. Given the cost of mailing an offer and the benefit of selling a policy (which depends on the customer), we could use the probability that the customer will buy a policy to choose which customers to mail an offer. However, since cost/benefit information is not available for this dataset, we cannot actually make the decisions to report profits. So, we just report the MSE for the different methods. We applied both naive Bayes and a linear kernel SVM to this dataset (we used the SvmFu package [69] with $C = 1$).

We show the MSE results for the raw naive Bayes and SVM scores and each calibration method in Table V.3. In order to obtain an MSE for SVM scores, we first re-scale them as explained in Section V.B. Again, by using each of the correction methods we are able to greatly improve the MSE for naive Bayes, but PAV performs slightly better than sigmoid fitting. The MSE for SVM scores is also reduced by the calibration methods. However, in this case sigmoid fitting is best. As shown in figure V.E.2, we also compared PAV to binning with bin sizes varying from 5 to 50 and found that PAV does slightly worse than binning with the optimal number of bins.

We also applied each method to the Adult dataset, which is available in the UCI ML Repository [10]. The prediction task is to determine whether a person makes over \$50K a year, given demographic information about the person. This dataset is also divided in a standard way into a training set (32561 examples) and

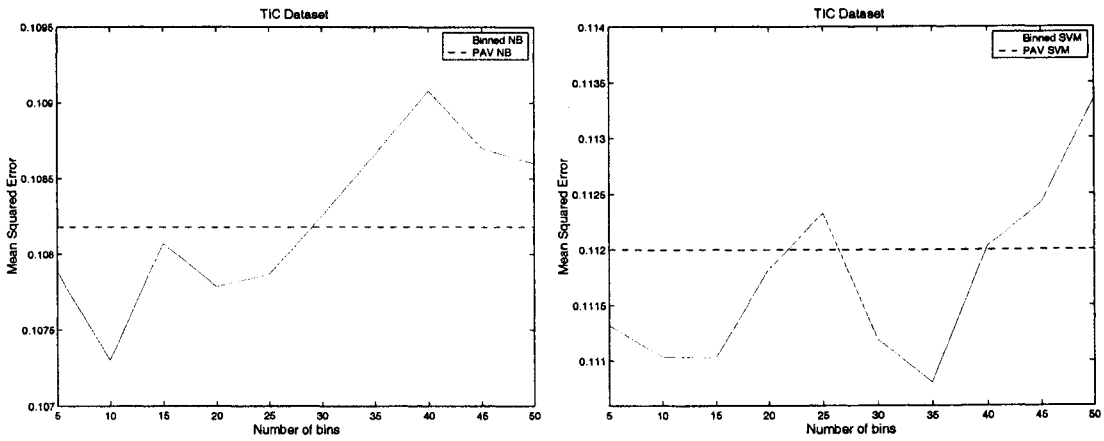


Figure V.E.2: Binning and PAV on the TIC dataset. The figure on the left shows results with Naive Bayes and the figure on the right shows results with SVM.

a test set (16281 examples). We apply both naive Bayes and SVM to this dataset, with no feature selection. For learning the SVM classifier, we use the SvmFu package [69] and, as done by Platt [63], use a linear kernel SVM ($C = 0.01$) with discretized features.

Table V.4 shows MSE and error rates for this dataset. The error rate is calculated by classifying x as positive if $\hat{P}(c|x) > 0.5$, where belonging to c indicates that x has income greater than \$50K. Note that by calibrating the naive Bayes scores, we reduce the error rate. This happens because 0.5 is not as good a threshold for the raw scores, as it is for the calibrated scores. However, with SVM the error rate is slightly increased when we apply the correction methods. This indicates that although the SVM scores are uncalibrated, the threshold used for classification is optimal. When the calibration methods are used, the error rate is increased because the refinement of the classifier is slightly reduced.

Surprisingly, even though the shape of the function mapping SVM scores to empirical probability estimates has a distinctive sigmoidal shape (Figure V.C.1), the PAV method performs slightly better than the sigmoid fitting method.

In figure V.E.3, we compare PAV to binning with bin sizes varying from 10

	MSE		Error Rate	
Method	Training	Test	Training	Test
NB	0.25112	0.25198	0.17100	0.17321
Sigmoid NB	0.21530	0.21515	0.15270	0.15190
PAV NB	0.20312	0.20452	0.14665	0.14831
SVM	0.28719	0.28684	0.15190	0.14968
Sigmoid SVM	0.20980	0.20962	0.15156	0.14993
PAV SVM	0.20815	0.20924	0.15115	0.15113

Table V.4: MSE and error rate on the Adult dataset.

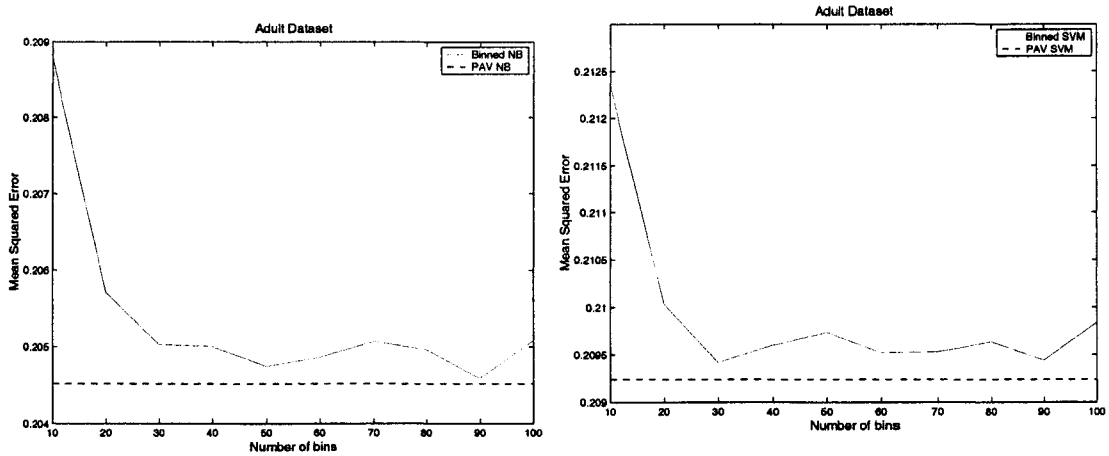


Figure V.E.3: Binning and PAV on the Adult dataset. The figure on the left shows results with Naive Bayes and the figure on the right shows results with SVM.

to 100 for both naive Bayes and SVM, and found that binning is always worse than PAV. This indicates that, for this dataset, by using a fixed number of examples per bin we cannot accurately model the mapping from SVM and naive Bayes scores into calibrated probability estimates.

V.E.2 Multiclass problems

The first multiclass dataset we consider is Pendigits, available in the UCI ML Repository [10]. It consists of 7494 training examples and 3498 test examples of pen-written digits (10 classes). The digits are represented as vectors of 16 attributes which are integers ranging from 0 to 100.

Method	MSE	Error Rate
NB Normalization	0.0326	0.1672
NB Least-Squares	0.0319	0.1672
NB Coupling	0.0304	0.1715
PAV NB Normalization	0.0241	0.1498
PAV NB Least-Squares	0.0260	0.1498
PAV NB Coupling	0.0260	0.1512
BNB Normalization	0.0163	0.0963
BNB Least-Squares	0.0164	0.0958
BNB Coupling	0.0160	0.1023
PAV BNB Normalization	0.0150	0.0946
PAV BNB Least-Squares	0.0150	0.0946
PAV BNB Coupling	0.0149	0.0935

Table V.5: MSE and error rate on Pendigits (test set).

For these experiments, we use a one-against-all code matrix. We use both naive Bayes and boosted naive Bayes as the binary learners, and apply PAV to calibrate the scores. As we mentioned in Section V.D there are two methods for combining binary probability estimates into multiclass probability estimates for arbitrary code matrices: least-squares and coupling. For one-against-all, however, there is another possible method: normalization. Because in this case each binary classifier i outputs an estimate of $P(c_i|x)$, we can simply normalize these estimates to make them sum to 1.

Table V.5 shows MSE and error rate when we apply each of the methods to naive Bayes, PAV naive Bayes, boosted naive Bayes and PAV boosted naive Bayes. When we calibrate the probability estimates before combining them using any of the methods, both the MSE and the error rate are lower than when we use raw scores. However, it is not clear which of the methods for combining the binary estimates is to be preferred. When the calibrated estimates are used it makes less difference which method is used. For this reason, we recommend using simple normalization for one-against-all, which is the simplest method.

The second multiclass dataset we use is 20 Newsgroups, which was collected and originally used by Lang [50]. It contains 19,997 text documents evenly

Method	MSE	Error Rate
NB Norm	0.01625 (± 0.00049)	0.15836 (± 0.0067)
NB LS	0.01720 (± 0.00045)	0.15530 (± 0.0064)
NB Coup	0.01585 (± 0.00041)	0.16066 (± 0.0075)
PAV NB Norm	0.01220 (± 0.00038)	0.15305 (± 0.0060)
PAV NB LS	0.01419 (± 0.00029)	0.15299 (± 0.0057)
PAV NB Coup	0.01415 (± 0.00029)	0.15422 (± 0.0060)

Table V.6: MSE and error rate on 20 Newsgroups.

distributed across 20 classes. Because there is no standard training/test split for this dataset, we randomly select 80% of documents per class for training and 20% for testing. We conduct experiments on 10 training/test splits and report mean and standard deviation.

Previous research [68] found that one-against-all performed as well as other code matrices for this dataset in terms of error rate, so we again restrict our experiments to one-against-all. We calibrate the naive Bayes scores using PAV, and apply each of the methods for obtaining multiclass probability estimates to both the raw naive Bayes scores and the PAV scores. Table V.6 shows MSE and error rates for each method. We see that by applying PAV to the binary naive Bayes scores, we can significantly reduce the MSE and slightly improve the error rate. The lowest MSE is achieved when we first calibrate the scores using PAV and then use normalization to obtain multiclass probability estimates.

V.F Conclusions

We have presented simple and general methods for obtaining accurate class membership probability estimates for two-class and multiclass problems, using binary classifiers that output ranking scores. We have demonstrated experimentally that our methods work well on a variety of data-mining domains and for different classifier learning methods.

For two-class problems, we recommend using the PAV algorithm to learn a mapping from ranking scores to calibrated probability estimates. For multiclass

problems, we first separate the problem into a number of binary problems, calibrate the scores from each binary classifier using PAV and combine them to obtain multiclass probabilities. We show experimentally that by calibrating the binary scores we can improve substantially the calibration of the multiclass probabilities obtained using one-against-all, the simplest way of breaking a multiclass problem into binary problems.

For many domains, however, using more sophisticated code matrices can yield better results, at least in terms of error rate [1]. Although we only conducted experiments using one-against-all, our method is applicable to arbitrary code matrices. More experiments are necessary to determine the best method for combining the binary probability estimates in the general case. One open question for future research is how to design an optimal code matrix for obtaining accurate class membership probability estimates.

Acknowledgments

The text of this chapter, in part, is a reprint of the material as it appears in the *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* [96]. The dissertation author was the primary author, and the co-author listed in this publication directed and supervised the research which forms the basis for this chapter.

Chapter VI

Sample selection bias

One of the most common assumptions in the design of learning algorithms is that the training data consist of examples drawn independently from the same underlying distribution of examples about which the model is expected to make predictions. In many real-world applications, however, this assumption is violated because we do not have complete control over the data gathering process.

For example, suppose we are using a machine learning method to induce a model that predicts what are the side-effects of a treatment for a given patient. Because the treatment is not given randomly to individuals in the general population, the available examples are not a random sample from the population. Similarly, suppose we are learning a model to predict the presence/absence of an animal species given the characteristics of a geographical location. Since data gathering is easier in certain regions than others, we would expect to have more data about certain regions than others.

In both cases, even though the available examples are not a random sample from the true underlying distribution of examples, we would like to learn a predictor from the examples that is as accurate as possible for this distribution. Furthermore, we would like to be able to estimate its accuracy for the whole population using the available data.

This problem has received a great deal of attention in econometrics, where

it is called sample selection bias. There it appears mostly because data are collected through surveys. Very often people that respond to a survey are self-selected, so they do not constitute a random sample of the general population. As we saw in Chapter III (Section III.D.1), Heckman [43] has developed a procedure for correcting sample selection bias. The key insight in Heckman's work is that if we can estimate the probability that an observation is selected into the sample, we can use this probability estimate to correct the model. The drawback of his procedure is that it is only applicable to linear regression models.

Also, in statistics, the related problem of missing data has been considered [54]. However, those methods are generally concerned with cases in which some of the features of an example are missing, and not with cases in which whole examples are missing.

In this chapter, we address the sample selection bias in the context of learning and evaluating classifiers. In Section VI.A we formally define the sample selection bias problem in machine learning terms. Then, in Section VI.B we present a new categorization of learning methods that is useful for characterizing their behavior under sample selection bias and study how a number of well-known classifier learning methods are affected by sample selection bias.

In Section VI.C, we present a bias correction method based on estimating the probability that an example is selected into the sample and using rejection sampling to obtain unbiased samples of the correct distribution. This method bears resemblance to the cost-sensitive learning methods by example weighting presented in Chapter IV and to weighting methods used in statistics for missing data [54]. It can be used both for learning classifiers and, more importantly, for evaluating a classifier using a biased sample.

VI.A Definition

Standard classifier learning algorithms (implicitly or explicitly) assume that we have examples (x, y), each drawn independently from a distribution D with domain $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is the feature space and $\mathcal{Y}$ is a (discrete) label space.

Here, we assume that examples (x, y, s) are drawn independently from a distribution D with domain $\mathcal{X} \times \mathcal{Y} \times \mathcal{S}$ where X is the feature space, Y is the label space and S is a binary space. The variable s controls the selection of examples (1 means the example is selected, 0 means the example is not selected). We only have access to the examples that have $s = 1$, which we call the selected sample. If the selected sample (ignoring s) is not a random sample of D we say that the selected sample is biased.

There are four cases worth considering for the dependence of the selection variable s on the example (x, y) [1]:

1. If s is independent of x and independent of y, the selected sample is not biased, that is, the examples (x, y, s) which have $s = 1$ constitute a random sample from D (ignoring s).

2. If s is independent of y given x (that is $P(s|x, y) = P(s|x)$), the selected sample is biased but the biasedness only depends on the feature vector x.

3. If s is independent of x given y (that is $P(s|x, y) = P(s|y)$), the selected sample is biased but the biasedness depends only on the label y. This corresponds to a change in the prior probabilities of the labels. This case has been studied in the machine learning literature and there are methods for correcting this type of bias [28, 9].

4. If no independence assumption holds between x, y and s, the selected sample is biased and we cannot hope to learn a mapping from features to labels using the selected sample, unless we have access to an additional feature vector

[1] In the statistics literature on missing data [54], cases (1), (2) and (4) are known as missing completely at random (MCAR), missing at random (MAR) and not missing at random (NMAR), respectively.

x_s that controls the selection (that is, $P(s|x_s, x, y) = P(s|x_s)$) for all the examples (even for the ones that have $s = 0$).

In econometrics, the usual assumption is (4) because the goal is to estimate the parameters of a model for y that reflects the true dependence of y on x. Any feature variable that only affects the selection should not be included in x (and it is included in x_s, instead). In machine learning, this is not a concern, because we are mostly interested in maximizing accuracy and not in obtaining the "correct" parameters for a model.

For this reason, we argue that the most important sample selection bias case in the practice of classifier learning is case (2). In order to make the condition $P(s|x, y) = P(s|x)$ true in practice, the input to the classifier x has to include all the variables that affect the sample selection. For example, in the medical treatment case, we need to include in x the variables about the patients that the doctors use to decide who gets the treatment (even if they do not affect the side-effects of the treatment directly).

Even if this assumption is not true in practice (either because we do not have access to all the variables that control the selection or because it truly depends directly on y), assuming case (2) is more realistic than the usual assumption of case (1). In the rest of this chapter, sample selection bias will refer to case (2).

VI.B Learning under sample selection bias

We can separate existing classifier learners into two categories:

- **local:** the output of the learner depends only on $P(y|x)$.
- **global:** the output of the learner depends on both $P(x)$ and $P(y|x)$.

The names "local" and "global" were chosen because $P(x)$ is a global distribution over the entire input space, while $P(y|x)$ is a local distribution, for each value of x.

Local learners are not affected by sample selection bias because

$$P(y|x, s = 1) = P(y|x)$$

while global learners are affected because the bias changes $P(x)$.

Although this categorization is very simple theoretically, it is not straightforward to classify existing learning methods into it. Below, we study analytically and experimentally how sample selection bias affects different types of classifiers learning methods, including Bayesian classifiers, logistic regression, support vector machines and decision trees.

VI.B.1 Bayesian classifiers

Bayesian classifiers compute posterior probabilities $P(y|x)$ using Bayes' rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

where $P(x|y)$, $P(y)$ and $P(x)$ are estimated from the training data. An example x is classified by choosing the label y that yields the highest posterior probability $P(y|x)$.

We can easily show that Bayesian classifiers are not affected by sample selection bias. To see this is true, note that by using the biased sample as training data, we are estimating $P(x|y, s = 1)$, $P(x|s = 1)$ and $P(y|s = 1)$ instead of estimating $P(x|y)$, $P(y)$ and $P(x)$. However, when we substitute these estimates into the equation above and apply Bayes' rule again, we see that we still obtain the desired posterior probability $P(y|x)$:

$$\frac{P(x|y, s = 1)P(y|s = 1)}{P(x|s = 1)} = \frac{\frac{P(y|x,s=1)P(x|s=1)}{P(y|s=1)}P(y|s = 1)}{P(x|s = 1)} = P(y|x, s = 1) = P(y|x)$$

since we are assuming that y and s are independent given x. Note that even though the estimates of $P(x|y, s = 1)$, $P(x|s = 1)$ and $P(y|s = 1)$ are different from the estimates of $P(x|y)$, $P(x)$ and $P(y)$, the differences cancel out. Therefore, bayesian learners are local learners.

In practice, we have a limited amount of training examples available to estimate $P(y|x)$. Compared to a random sample of the same size, the biased sample contains more examples in parts of the feature space where $P(s = 1|x)$ is high and less examples where $P(s = 1|x)$ is low. This will lead to estimates of $P(y|x)$ with lower variance where $P(s = 1|x)$ is high and with higher variance where $P(s = 1|x)$ is low. However, as long as $P(s = 1|x)$ is greater than zero for all x, as we increase the sample size, the results on a selected sample will asymptotically approach the results on a random sample.

Naive Bayes

In practical Bayesian learning, we often make the assumption that the features are independent given the label y, that is, we assume that

$$P(x_1, x_2, \ldots, x_n|y) = P(x_1|y)P(x_2|y)\ldots P(x_n|y).$$

This is the so-called naive Bayes assumption.

With naive Bayes, unfortunately, the estimates of $P(y|x)$ obtained from the biased sample are incorrect. In this case the desired posterior probability $P(y|x)$ is estimated as

$$\frac{P(x|y, s = 1)P(y|s = 1)}{P(x|s = 1)}$$

$$= \frac{P(x_1, x_2, \ldots, x_n|y, s = 1)P(y|s = 1)}{P(x|s = 1)}$$

$$= \frac{P(x_1|y, s = 1)P(x_2|y, s = 1)\ldots P(x_n|y, s = 1)P(y|s = 1)}{P(x|s = 1)}$$

which is different (even asymptotically) from the estimate of $P(y|x)$ obtained with naive Bayes without sample selection bias. We cannot simplify this further because there are no independence relationships between each x_i, y and s. Therefore, naive Bayes learners are global learners.

VI.B.2 Logistic regression

In logistic regression, we use maximum likelihood to find the parameter vector β of the following model [2].

$$P(y=1|x) = \frac{1}{1+\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}$$

With sample selection bias we will instead fit the parameters of

$$P(y=1|x,s=1) = \frac{1}{1+\exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}$$

However, because we are assuming that y is independent of s given x we have that

$$P(y=1|x,s=1) = P(y=1|x).$$

Thus, logistic regression is not affected by sample selection bias, except for the fact that the number of examples is reduced. Asymptotically, as long as $P(s=1|x)$ is greater than zero for all x, the results on a selected sample will approach the results on a random sample. In fact, this is true for any learning method that models $P(y|x)$ directly. These are all local learners.

Figure VI.B.1 illustrates the effect of sample selection bias on logistic regression for synthetically generated data, where x is one-dimensional. The graph on the left-hand size shows 1000 points where the x value is chosen uniformly between -10 and 10 and the y value is drawn with probabilities calculated using a logistic function (β_0=3 and β_1=2). The curve is the logistic function obtained through maximum likelihood using the plotted points. The dashed line is the separator between the two classes. The graph on the right-hand side shows a selected sample of the points, where the probability of each point being selected is proportional to its x value. We also show the logistic function obtained through maximum-likelihood using the selected points. We can see in the graphs that although the selected sample contains many less points on the negative side than the original sample, the estimated curve and the resulting separator are the same.

[2] We show the two-class version; for the multiclass version see [41]

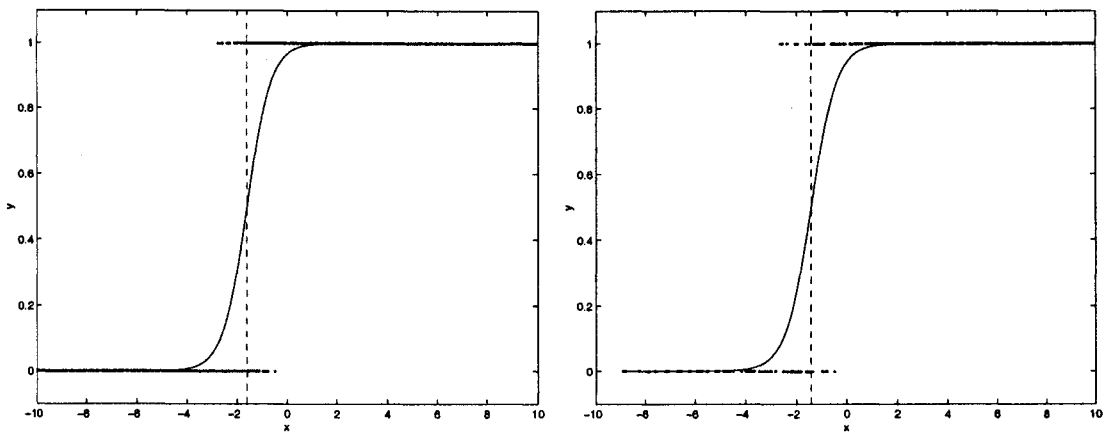


Figure VI.B.1: Logistic regression is unaffected by sample selection bias.

VI.B.3 Decision tree learners

Decision tree learners such as C4.5 [66] and CART [15] split the input space x in a recursive, top-down manner. Figure VI.B.2 shows an example of a decision tree for a direct marketing problem. Each branch is a test on the value of one the features. For discrete features, the tree branches into nodes corresponding to each of the possible feature values. For real-value features, the tree branches into two nodes corresponding to some threshold on the feature. To predict the class of a new example, we work down the tree, at each node choosing the appropriate branch by comparing the example with the values of the variable being tested for that node [38].

The splitting criteria used by different decision tree learners to grow a tree vary, but they are all based on calculating the impurity of the nodes after the split. For example, CART uses the GINI index

$$\text{GINI(t)} = 1 - \sum_{y} P(y|t)$$

where $p(y|t)$ is the relative frequency of class y at node t. The GINI index is at its maximum when the examples are equally distributed among the classes and at its minimum when all the examples belong to the same class. For each possible split

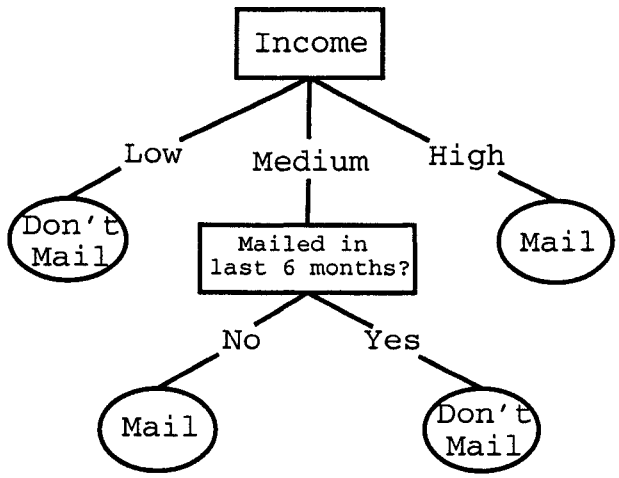


Figure VI.B.2: A decision tree.

of the data, CART calculates

$$\sum_{i=1}^{k} \frac{n_i}{n} \text{GINI}(i)$$

where k is the number of nodes induced by the split.

C4.5 uses information (or entropy) instead of the GINI index, which is given by

$$\text{INFO(t)} = -\sum_{y} P(y|t) \log P(y|t)$$

where $P(y|t)$ is the relative frequency of class y at node t. Like the GINI index, INFO is at its maximum when the examples are equally distributed among the classes and at its minimum when all the examples belong to the same class.

Because the splitting criteria are dependent on $P(y|t)$, where t is a test on only one of the feature values, and, in general,

$$P(y|t, s = 1) \neq P(y|t),$$

the splits chosen by the learners are sensitive to sample selection bias. Thus, decision tree learners are global learners.

Even though the choice of each split is affected by sample selection bias, the final tree may not be. If the tree splits the input space into small enough

rectangles, the proportion of examples of class y in each rectangle will be an unbiased estimate of the probability of class y for that rectangle, even under sample selection bias. Thus, the majority class will be the correct label for new examples that fall into it.

VI.B.4 Support vector machines

In its basic form, the support vector machine (SVM) algorithm [86, 44] learns the parameters a and b describing a linear decision rule

$$h(x) = \text{sign}(a \cdot x + b),$$

whose sign determines the label of an example, so that the smallest distance between each training example and the decision boundary (called the margin) is maximized.

Given a sample of examples (x_i, y_i), where $y_i \in \{-1, 1\}$, it accomplishes margin maximization by solving the following optimization problem:

$$\begin{aligned} &\text{minimize: } V(a, b) = \tfrac{1}{2} a \cdot a \\ &\text{subject to: } \forall i: \quad y_i[a \cdot x_i + b] \geq 1 \end{aligned}$$

The constraint requires that all examples in the training set are classified correctly. Thus, sample selection bias will not systematically affect the output of such this optimization, assuming that the selection probability $P(s = 1|x)$ is greater than zero for all x.

Figure VI.B.3 illustrates the effect of sample selection bias on SVM for synthetically generated data, where x is one-dimensional. The graph on the left-hand size shows 500 points for each of two classes, generated from two different two-dimensional gaussians. The line is the maximal margin separator. The graph on the right-hand side shows a selected sample from these points where the probability of each point being selected is proportional to its x value. We also show maximal marginal separator using the selected points. We can see in the graphs

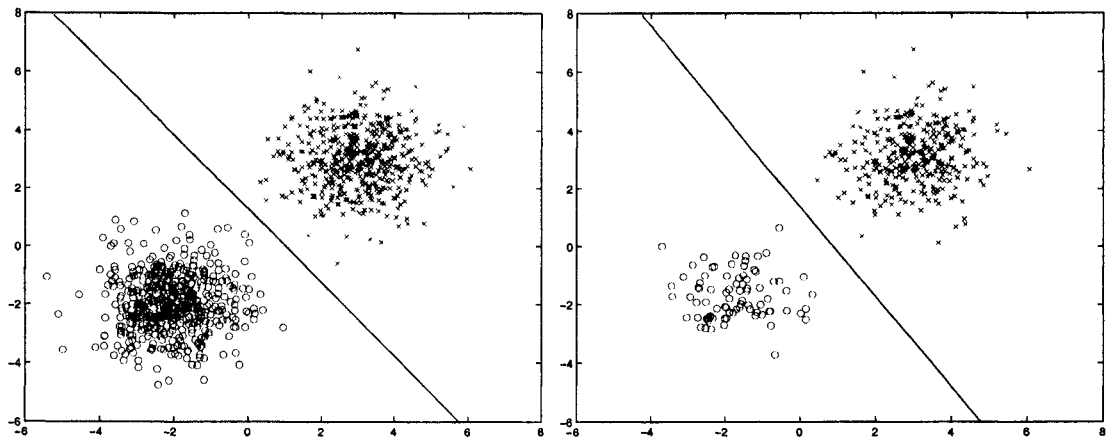


Figure VI.B.3: SVM for separable data is unaffected by sample selection bias.

that although the selected sample contains many less points on the negative side than the original sample, the resulting separator is not significantly altered.

In practice, a decision rule that classifies all the examples correctly may not exist because of overlap of the classes. To allow for the possibility of misclassified examples, one introduces slack variables $\xi_i > 0$ for each example (x_i, y_i). This is called a soft margin support vector machine classifier [72]. The optimization problem is changed to

$$\begin{aligned} &\text{minimize: } V(a,b,\xi) = \tfrac{1}{2}a \cdot a + C\sum_{i=1}^{n} \xi_i \\ &\text{subject to: } \forall i: \quad y_i[a \cdot x_i + b] \geq 1 - \xi_i, \ \xi_i > 0 \end{aligned}$$

If a training example lies on the wrong side of the decision boundary, the corresponding ξ_i is greater than 1. Therefore, $\sum_{i=1}^{n} \xi_i$ is an upper bound on the number of training errors. The factor C is a parameter that allows one to trade off training error and model complexity. We note that the algorithm can be generalized to non-linear decision rules by replacing inner products with a kernel function [86] in the formulas above.

Now, while sample selection bias does not affect the original SVM optimization, it does affect the soft margin optimization because it optimizes the sum of ξ_i values. By making regions of the feature space denser than others, sample

selection bias changes this sum and, with it, the decision boundary. Soft margin SVM is a global algorithm because changes in $P(x)$ will change the output.

VI.B.5 Experimental results

To verify the effects of sample selection bias experimentally, we apply Naive Bayes, logistic regression, C4.5 and SVMLight (soft margin) [45] to the Adult dataset (see Chapter V, section V.E.1 for information on this dataset). We assume that the original dataset is not biased and artificially simulate biasedness by generating a value for s for each example, such that s is correlated with one of the input features. When training, we only use the examples in the training set for which $s = 1$. When testing, we use all the examples in the test set, because we are interested in measuring the performance of the classifiers on the original distribution of examples.

Figure VI.B.4 shows the results of applying the different learners to the Adult dataset using unbiased and biased training sets of increasing size. For each size shown on the x-axis, we generated 50 unbiased samples from the original Adult training set. We also generated 50 biased samples by assigning s such that examples with feature **age** less than 30 are 9 times more likely to have $s = 1$ than examples with **age** more than 30. We trained the learners using each of the 50 samples (in both the biased and unbiased cases) and tested each of the models on the Adult test set, to obtain the mean and standard error of the error rate, as shown in the graphs.

In accordance with our analysis, for logistic regression, the difference in error rate between using biased or unbiased training sets goes down as we increase the size of the training set. Also, as expected, we see that naive Bayes is very sensitive to sample selection bias. The error rate using the biased sample goes up as we increase the number of training examples.

On the other hand, surprisingly, C4.5 performs very well under sample selection bias. This might be explained by the fact that even though the choice of

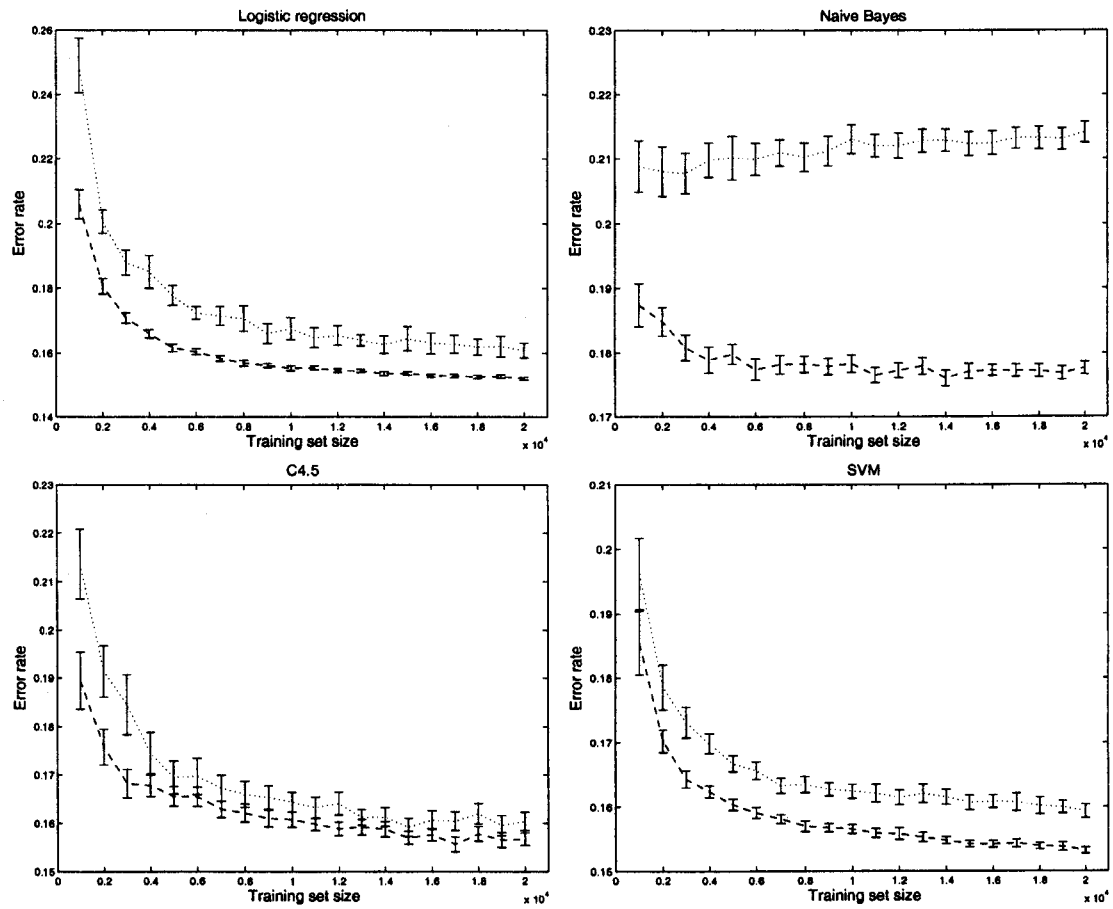


Figure VI.B.4: Error rate using biased (dotted) and unbiased (dashed) training sets. Each point indicates the mean error rate for a given sample size and the error bars show the standard error of the error rate. These were computed using 50 different training sets for each size.

splits is biased, the class estimates at the leaves are not. More experiments with different types of selection biases are necessary to understand the effect of sample selection bias on decision tree learners.

With SVM, we see that the error rate using the biased training set decreases as the training set sizes increases. However, the difference between the error rates using biased and unbiased samples does not decrease. This indicates that, asymptotically, SVM (with soft margin) is affected by sample selection bias.

VI.C Correcting sample selection bias

In the last section, we saw that some classifier learning methods are affected by sample selection bias, while others are not. In this section, we present a bias correction method that can be applied to any classifier learner, provided that we have a model for the selection probabilities $P(s = 1|x)$. The method works by correcting the distribution of examples through re-sampling and then applying the classifier learner to the corrected sample.It bears resemblance to weighting methods proposed in the statistics literature for missing data [54] and also to the cost-sensitive learning by example weighting methods presented in Chapter IV.

Classifier learners try to find h to minimize the expected value of loss function over the distribution of examples given by

$$E_{x,y\sim D}[l(h(x), y)].$$

The loss function is, in many cases, given by an indicator of error $I(h(x) \neq y)$, but we make the analysis more general by considering an arbitrary loss function (such as the one used in cost-sensitive learning).

Under sample selection bias, a classifier learner will minimize instead

$$E_{x,y,s\sim D}[l(h(x), y)|s = 1]$$

because only the examples with $s = 1$ are available to the learner.

Assume that we know the selection probabilities $P(s = 1|x)$ and that they are greater than zero for all x. Let $\hat{D}$ be a new distribution such that

$$\hat{D}(x, y, s) \equiv P(s = 1)\frac{D(x, y, s)}{P(s = 1|x)}.$$

where $P(s = 1) = \sum_{(x,y,s)\sim D} P(s = 1, x)$ is the overall selection probability.

The following theorem shows that if we change the distribution of examples from D to $\hat{D}$, we will obtain the desired expected value under sample selection bias.

Theorem VI.C.1. *(Bias Correction Theorem) For all distributions, D, for all classifiers, h, for any loss function $l = l(h(x), y)$, if we assume that $P(s = 1|x, y) = P(s = 1|x)$ (that is, s and y are independent given x) then*

$$E_{x,y\sim D}[l(h(x), y)] = E_{x,y,\sim\hat{D}}[l(h(x), y)|s = 1]$$

Proof.

$$\begin{aligned}
E_{x,y,s\sim\hat{D}}[l(h(x), y)|s = 1] &= \sum_{x,y} l(h(x), y)P_{\hat{D}}(x, y|s = 1) \\
&= \sum_{x,y} l(h(x), y)P_{\hat{D}}(x, y|s = 1) \\
&= \sum_{x,y} l(h(x), y)\frac{P_D(s = 1)}{P_D(s = 1|x)}P_D(x, y|s = 1) \\
&= \sum_{x,y} l(h(x), y)\frac{P_D(s = 1)}{P_D(s = 1|x)}\frac{P_D(s = 1|x, y)P_D(x, y)}{P_D(s = 1)} \\
&= \sum_{x,y} l(h(x), y)P_D(x, y) \\
&= E_{x,y\sim D}[l(h(x), y)]
\end{aligned}$$

□

The left-hand side ($E_{x,y\sim D}[l(h(x), y)]$) is the expected value that we would like to minimize but cannot directly under sample selection bias. The right-hand side ($E_{x,y,s\sim\hat{D}}[l(h(x), y)|s = 1]$) can be minimized as long as we can draw examples from $\hat{D}$.

As we have seen in Chapter IV obtaining a sample from a weighted distribution given a finite set of training examples is not completely straightforward. We have demonstrated that costing, a method based on rejection sampling, achieves the best results in practice. For this reason, we recommend using costing for sample selection bias correction, where instead of using misclassification costs as weights we use the selection ratio $P(s=1)/P(s=1|x)$ as a weight for each example.

Up to now, we have assumed that we know the selection probabilities $P(s=1|x)$. In practice, we would have to estimate these from data. If we have a sample $(x,s) \sim D$ (note that y is not necessary), we can use it to estimate these probabilities using a classifier learning method along with the calibration methods presented in Chapter V. Note that this is a two-class problem, since s takes values in $\{0,1\}$.

This assumes that we have unlabeled examples drawn from the true underlying distribution and can determine whether they are selected or not. This is a situation that is likely to occur in practice. For example, in medical treatment, we only know the outcome of the treatment (y) for patients x that were given the treatment ($s=1$). On the other hand, we can come up with examples of the form (x,s) that are drawn from the population as a whole.

VI.C.1 Evaluation under sample selection bias

In evaluation, for a given a classifier h, we would like to obtain an estimate of the loss of the classifier, given by

$$E_{x,y\sim D}[l(h(x),y)].$$

Usually this is done by applying the classifier to a set of test examples drawn from D and obtaining the empirical loss on the test examples

$$\frac{1}{m}\sum_{(x,y)} l(h(x),y)$$

where m is the number of available examples.

However, under sample selection bias, since we only see the examples for which $s = 1$, we instead obtain an estimate of

$$E_{x,y,s\sim D}[l(h(x), y)|s = 1],$$

which in general is not an unbiased estimate of the loss of the classifier.

As we have seen in Section VI.B, local learning methods are insensitive to sample selection bias. However, the evaluation step is always affected by sample selection bias because we are calculating an expected value over the whole input space (which is always "global"). Therefore, we argue that accounting for sample selection bias on the evaluation step is more important than accounting for sample selection bias during learning.

We can use the bias correction theorem (theorem VI.C.1) for evaluating a classifier if we have estimates of the selection probabilities $P(s = 1|x)$. We simply have to weigh each example by $P(s = 1)/P(s = 1|x)$ when calculating the expected loss on the biased test sample. Thus, the empirical loss is given by

$$\frac{1}{m}\sum_{(x,y,s)}\frac{P(s=1)}{P(s=1|x)}l(h(x),y)$$

$$= P(s=1)\sum_{(x,y,s)}\frac{l(h(x),y)}{P(s=1|x)}.$$

Again, in practice, we would have to estimate $P(s = 1|x)$ and $P(s = 1)$ from a set of examples.

VI.C.2 Example

To illustrate how the bias correction method works, we constructed an example using the KDD-98 competition dataset described in Chapter II. This example is artificial in the sense that we assume we know the selection probabilities $P(s = 1|x)$ and we enforce the selection of examples using these probabilities. By doing this, we can compare the estimates of the expectation obtained using the whole sample and using the selected sample (corrected and uncorrected).

The KDD-98 dataset contains information about persons who have made donations in the past to a particular charity. For the purpose of this example, we only need to look at two variables: income and amount. Income is a variable that takes values in $\{0, 1, 2, 3, 4, 5, 6, 7\}$ and indicates the different levels of income (from lower to higher). Amount is how much the person has donated (in dollars) in the last donation campaign. We only use examples of people that have donated in the last campaign.

In the notation of the theorem, income is x and amount is l. (We chose to side-step the classifier $h(x)$ and the label y and assume we have the loss l directly for each example).

Suppose that s is such that

$$P(S = 1|X = x) = \begin{cases} 0.3 & \text{if } x \in \{0, 1, 2, 3\} \\ 0.9 & \text{if } x \in \{4, 5, 6, 7\} \end{cases}$$

In this case, the overall probability of selection $P(s = 1)$ is 0.6.

The empirical estimate of the expected amount obtained by averaging the amounts of all the examples is 15.62. Because there is a positive correlation between income and donation amount, if we select the examples according to the probabilities above, we will overestimate the expected amount.

To demonstrate this experimentally we can assign s values for each example according to the probabilities above and calculate the empirical mean of l using only the examples that have $s = 1$. By repeating this for 1000 different random draws of s, i.e., 1000 different selected samples, we obtain the distribution of estimated expected values of Y seen in Figure VI.C.1. The vertical dashed line (on the left side) shows the estimated expected amount using the whole sample. The graph shows that, by using only the selected examples to estimate the expected value of l, we consistently overestimate it, as expected.

In contrast, Figure VI.C.2 shows the distribution of estimated expected values for l, when we use only the selected examples but apply the bias correction method. The distribution is centered near the value estimated from the whole

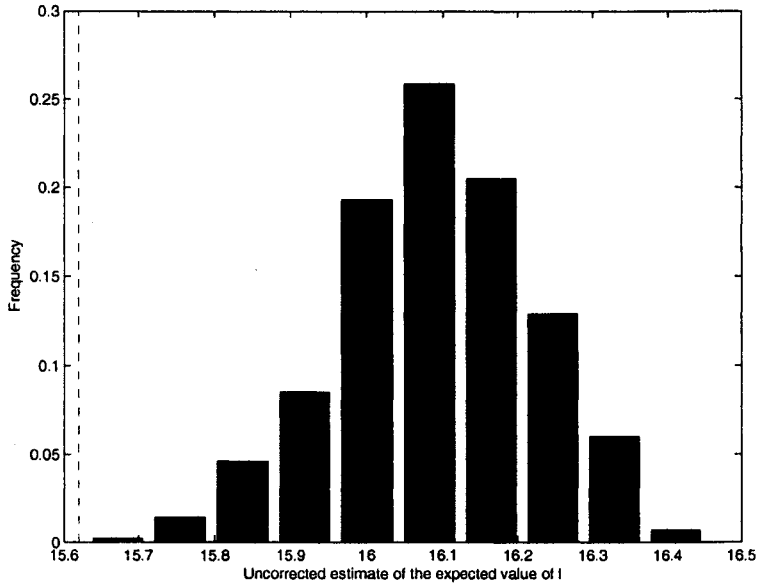


Figure VI.C.1: Distribution of the **uncorrected** estimates of expected amount (l) when different selected samples are used. The vertical dashed line shows the estimated expected value using the whole sample.

sample (and the mean is 15.62). Therefore, we can conclude that, in fact, the proposed method succeeds at correcting the bias. We note, however, that the variance is increased when we use the correction method (from 0.0170 to 0.0434).

In this case, we knew the selection probabilities, so we used them directly. In a more realistic case, these probabilities would have to be estimated from data.

VI.D Conclusions

In this chapter we have presented a formal definition of the problem of sample selection bias in classifier learning. By studying the behavior of different classifier learners under sample selection bias analytically and experimentally, we separated existing classifier learners into two categories:

- **local:** the behavior of these learners only depends on $P(y|x)$. Examples: logistic regression, SVM (without soft margin).

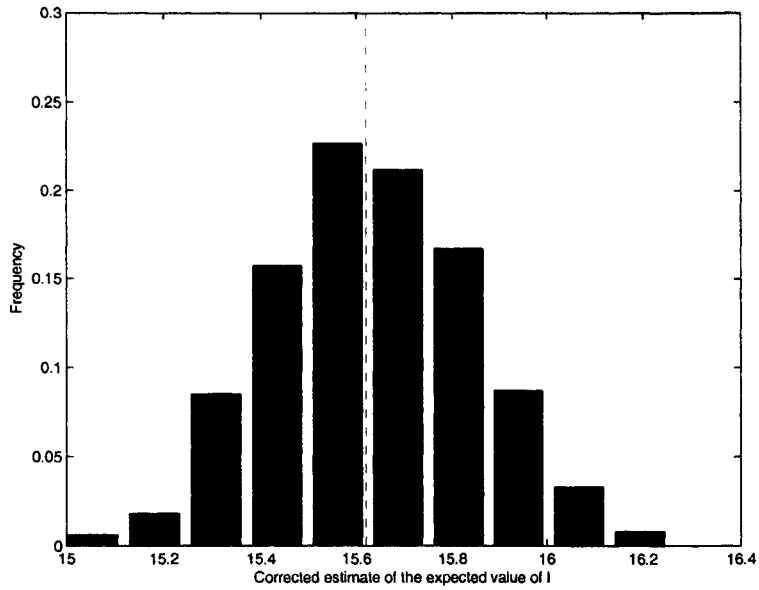


Figure VI.C.2: .

Distribution of the **corrected** estimates of expected amount (l) when different selected samples are used. The vertical dashed line shows the estimated expected value using the whole sample.

- **global:** the behavior of these learners depends on both $P(x)$ and $P(y|x)$. Examples: naive Bayes, SVM (with soft margin), decision tree learners.

While global learners are affected by sample selection bias, local learners are not. This is a new categorization that is different than the more usual categorization of learning methods into discriminative and generative learners [61]. As we have seen in Section VI.B.1, although generative (or Bayesian) methods model $P(x|y)$, $P(y)$ and $P(x)$, their behavior is generally independent of $P(x)$ (although this is not true for naive Bayes).

This categorization is also useful for characterizing situations in which we can learn from both labeled and unlabeled data, an area of research that has received some attention in recent years (see, for example, [80]). Clearly, global learners can take advantage of unlabeled data, while local learners cannot.

For global learners, we showed that we can still learn correctly under sample selection bias if we have data to estimate the selection probabilities $P(s = 1|x)$. Also, we showed how to evaluate a classifier using a biased sample and the estimates of the selection probabilities. The calibration methods presented in Chapter V can be used for obtaining these estimates using classifier learning methods.

Chapter VII

Reinforcement learning with traces

In this chapter we give an overview of reinforcement learning and formally define the policy mining problem using the Markov Decision Process (MDP) framework that is commonly used in reinforcement learning. We argue that the current reinforcement learning methodology is not suitable for the policy mining setting and present a new formulation that we call *reinforcement learning with traces*. This formulation does not require the availability of a simulator for the environment and, instead, uses a trace model that can be simulated with fixed sets of data collected offline. We show that for one-step MDPs, we can reduce reinforcement learning with traces to cost-sensitive learning with sample selection bias correction. For MDPs with arbitrary number of steps, we present a greedy iterative method that learns a classifier for each step. The policy obtained with this method is the approximately best possible local improvement over the arbitrary policy used for collecting the data. We also show how to evaluate a policy using a fixed set of data by using the sample selection bias correction methods presented in Chapter VI. Finally we present data mining applications that can benefit from this methodology and show experimental results using a data generator.

VII.A Reinforcement learning

In the reinforcement learning framework [78], an agent interacts with its environment by executing actions. The environment responds to those actions by presenting new situations and rewards to the agent. At each time step t, the environment is in some state s, the agent takes one of several actions a, receives a finite reward r, and the environment makes a transition to another state s'.

If the probability distributions of the reward r and the next state s' only depend on the current state s and action a, we say that the environment satisfies the Markov property. Many environments satisfy this property and are called Markov decision processes (MDPs).

In this case, we can completely specify the environment by specifying a tuple $(S, D, A, \{P_{sa}\}, \gamma, \{R_{sa}\})$, where S is a set of states; D is the initial-state distribution; A is a set of actions; $\{P_{sa}\}$ are the transition probabilities, with P_{sa} giving the next-state distribution when action a is executed in state s; $\gamma \in [0, 1]$ is a discount factor; and $\{R_{sa}\}$ are the reward distributions, with R_{sa} giving the reward distribution when action a is executed in state s.

The environment starts in an initial state s_0 drawn from the initial-state distribution and the learner repeatedly takes actions until a final state s_n is reached. This results in a sequence of states $\{s_t\}_{t=0}^{n}$, actions $\{a_t\}_{t=0}^{n}$, and rewards $\{r_t\}_{t=0}^{n}$, that is called an episode.

When choosing actions, the agent follows a policy that can be represented by a mapping $\pi(s)$, from states to actions. The value of a policy π is the expected discounted sum of rewards obtained when π is executed, which is given by

$$V(\pi) = E_{s_0 \sim D, \pi}\left[\sum_{t=0}^{n} \gamma^t r_t\right],$$

where $E_{s_0 \sim D, \pi}$ denotes that the initial state is drawn from the initial-state distribution and that we use policy π to choose the actions.

The optimal policy π^* is the policy that maximizes $V(\pi)$. Reinforcement learning (RL) methods attempt to learn the optimal policy by interaction with the

environment or with a simulator of the environment. Current RL methods can be divided into two categories: indirect and direct methods.

VII.A.1 Indirect Methods

Indirect methods first estimate the function $Q^*(s,a)$ that gives the expected value of executing action a in state s under the optimal policy, that is,

$$Q(s,a) = E_{\pi^*}\left[\sum_{t=0}^{n} \gamma^t r_t \middle| s_0 = s, a_0 = a\right]$$

where E_{π^*} denotes the expectation with respect to the optimal policy π^* which is used to define the actions taken in all states except the initial s_0. The optimal policy can be obtained from $Q(s,a)$ by choosing the action that maximizes $Q(s,a)$ at each state, that is,

$$\pi^*(s) = \text{argmax}_a Q(s,a)$$

When the set of possible states and actions is finite, it is possible to learn $Q(s,a)$ using online iterative algorithms such as Q-learning [90] and sarsa [71]. These methods start with an estimate of $Q(s,a)$ for every s and a and update the estimate after each action is executed. Provided that every action in every reachable state is executed infinitely often, they are guaranteed to probabilistically converge to the optimal value function [91].

However, in most practical applications, the state space is infinite or prohibitively large. In this case, instead of representing the value function explicitly as a look-up table, we can represent it as a parameterized function of the state-action pair. Given examples of the form $(\langle s,a\rangle, Q(s,a))$ function approximation methods such as linear regression [82], neural networks [81, 98] or decision trees [89] are used to create a mapping from state-action pairs to Q-values. The problem with this approach is that little is known about convergence guarantees and error bounds for the policy derived from the function approximator.

VII.A.2 Direct Methods

Direct methods do not estimate an intermediary function from which to derive the policy. Instead, they search for a good policy in a restricted class of policies. Given a fixed class of policies Π, the goal is to find a policy $\pi \in \Pi$ that is the best policy in the class. The best possible value achievable in class Π is given by

$$V^*(\Pi) = \sup_{\pi \in \Pi} V(\pi).$$

Direct methods attempt to find a policy $\hat{\pi} \in \Pi$ such that $V(\hat{\pi})$ is as close as possible to $V^*(\Pi)$.

Different direct methods vary on the optimization procedure used for searching the space of policies and on the method used for estimating the value of a policy. If the action space is continuous and $\Pi = \{\pi_\theta | \theta \in R^m\}$ is a smoothly parameterized family of policies it is possible to use gradient descent methods for searching the space of policies [5, 79]. Otherwise, optimizations procedures such as the downhill simplex method [53] and differential evolution [76] can be used. Note that some of these optimization procedures can be implemented using only comparative information of the form "Is policy π_A better than policy π_B?" [77].

All the existing methods for estimating the value of a policy or for comparing the values of two policies assume that we have access to a simulator of the underlying MDP. Most methods assume access to a stochastic function that takes as input any state-action pair $\langle s, a \rangle$ and outputs the next state s' and the reward r according to the state transition and reward probabilities of the MDP.

Other methods make even stronger assumptions. For example, the Pegasus method [60] assumes that we have a function $g : S \times A \times [0,1]^d \mapsto S$, such that for any fixed state-action pair $\langle s, a \rangle$ and p distributed uniformly in $[0,1]^d$, $g(s, a, p)$ gives the distribution of possible transition states when action a is executed in state s (they assume that the rewards are deterministic). This function is used to obtain the value of a policy for a scenario (i.e. a given initial state and values of p).

VII.B The policy mining setting

In the most general setting of policy mining, we have a set of examples each of the form $(\langle s_0, a_0, r_0\rangle, \langle s_1, a_1, r_1\rangle, \ldots, \langle s_T, a_T, r_T\rangle)$, where $T \geq 1$. Each example describes a fixed-length episode of interaction with the same unknown MDP, where the actions are chosen according to an unknown and, in general, non-deterministic, policy. We assume that each initial state is an i.i.d. sample from the initial-state distribution. We call these examples "traces", because they are traces of the execution of a policy on an MDP. We now want to learn a good policy for the MDP exclusively from these traces. We also want to be able to evaluate the policy to convince ourselves that it performs well before deploying it in the real-world.

One possible solution is to try to learn a simulator of the underlying MDP using the data and then use direct reinforcement learning methods to learn a policy. This requires learning models for predicting the transition probabilities and the reward distribution given a state-action pair. Since for most practical applications the state space is infinite, we would have to use function approximation methods to learn the models. The drawback of this approach is that modeling the MDP may be very difficult and, as a consequence, the policy that we derive from the simulator is likely to be suboptimal for the true MDP. Furthermore, it could be difficult to quantify how suboptimal it is.

We could also use indirect methods and learn a Q-value function using the data. This is the approach taken by Pednault et al. [62], called batch reinforcement learning. However, as we saw in Section VII.A.1 indirect methods with function approximation are not guaranteed to yield a good policy. Another problem is that we still need to evaluate the policy learned. The solution used by Pednault et al. is to learn a simulator of the underlying MDP and then run simulations using the policy. But, again, the simulator is only a crude approximation of the MDP and it may behave very differently from the original MDP.

Here, we present a direct reinforcement learning method that does not require a simulator and, instead, uses a fixed set of episodes to learn the policy. We call this method reinforcement learning with traces, because all that is provided to the method are traces of the execution of a policy. In the next section, we show that in the case of one-step MDPs ($T = 1$), we can solve reinforcement learning with traces by reducing it to a cost-sensitive classifier learning problem with sample selection bias correction.

VII.C One-step reinforcement learning with traces

We assume that we have m training examples (x, y, r) drawn from a joint distribution D with domain $\mathcal{X} \times \mathcal{Y} \times \mathcal{R}$ where $\mathcal{X}$ is an (arbitrary) state space, $\mathcal{Y}$ is a (discrete) action space and $\mathcal{R}$ is (nonnegative, real) reward space.

We can view these examples as being generated by repeatedly executing a stochastic training policy on a one-step MDP and recording traces of the execution in the form of state-action-reward triples, where the reward of executing action y in state x is is given by a stochastic function $R : \mathcal{X} \times \mathcal{Y} \to [0, \infty]$ (that is $r \sim R(x, y)$).

Our goal is to approximate the optimal policy for this MDP, i.e., a function $h : \mathcal{X} \to \mathcal{Y}$ that maximizes the expected value of the reward given by

$$E_{x \sim D}[R(x, h(x))] \tag{VII.C.1}$$

using only the available examples.

Standard classifier learners try to find H to maximize the accuracy

$$\frac{1}{m} \sum_{(x,y)} I(H(x) = y)$$

but, according to the translation theorem (Proposition IV.A.1), can be made to maximize

$$\frac{1}{m} \sum_{(x,y,w)} w I(H(x) = y). \tag{VII.C.2}$$

The following theorem shows that the expectation in (VII.C.1) can be rewritten in a way that allows us to use a classifier learner that maximizes (VII.C.2) to learn the policy h.

Theorem VII.C.1. *For all distributions, D, for any deterministic function, $h : \mathcal{X} \to \mathcal{Y}$ and for any stochastic function $R : \mathcal{X} \times \mathcal{Y} \to [0, \infty]$, if we assume that $P(y|x) > 0 \ \forall x, y$ then*

$$E_D[R(x, h(x))] = E_D\left[\frac{r}{P(y|x)} I(h(x) = y)\right]$$

Proof.

$$\begin{aligned}
E_D\left[\frac{r}{P(y|x)} I(h(x) = y)\right] &= E_{x,y,r\sim D}\left[\frac{R(x,y)}{P(y|x)} I(h(x) = y)\right] \\
&= E_D\left[\frac{R(x,y)}{P(y|x)} \middle| h(x) = y\right] P(h(x) = y) \\
&= E_D\left[\frac{R(x,h(x))}{P(h(x) = y|x)} \middle| h(x) = y\right] P(h(x) = y) \\
&= \sum_x \frac{R(x,h(x))}{P(h(x) = y|x)} P(x|h(x) = y) P(h(x) = y) \\
&= \sum_x \frac{R(x,h(x))}{P(h(x) = y|x)} P(x, h(x) = y) \\
&= \sum_x R(x,h(x)) P(x) \\
&= E_D[R(x,h(x)]
\end{aligned}$$

□

From this theorem, it follows that

$$\frac{1}{m} \sum_{(x,y,r)} \frac{r}{P(y|x)} I(h(x) = y), \tag{VII.C.3}$$

is an unbiased empirical estimate of the value of policy h. Thus, if we know $P(y|x)$, that is, the probability that action y is executed in state x by the training policy, we can use a classifier learner to learn the policy from the examples. Looking back at (VII.C.2) we see that we simply have to weigh each example (x, y, r) by $\frac{r}{P(y|x)}$.

Note that the theorem holds only if $\forall x, y\ P(y|x) > 0$, that is, in order to guarantee convergence to the optimal policy, we require that the training policy have non-zero probability of executing each action in each state. However, the reduction degrades gracefully even when this is not the case if we define that

$$\frac{I(h(x) = y)}{P(y|x)} = 0$$

when $I(H(x) = y) = 0$ and $P(y|x) = 0$. In this case, it is easy to see that the reduction will converge to the optimal policy for an MDP that is identical to the original one, except that action y is not allowed in state x.

This theorem can be seen as a combination of the translation theorem for cost-sensitive learning (Chapter IV, Theorem IV.A.1) and the bias correction theorem (Chapter VI, Theorem VI.C.1). This demonstrates that in policy mining it is crucial to account both for costs (or, in this case, rewards) and for the sample selection bias related to the use of a training policy that is not completely random.

In Chapter IV we showed that learning from a weighted distribution of examples is not straightforward with many classifier learners but that costing, a method based on rejection sampling, achieves good results in practice. For this reason, we recommend using costing here, where instead of using misclassification costs as weights we use the ratio $\frac{r}{P(s=1|x)}$ as a weight for each example (x, y, r). Another option is to use the transparent box methods, with learners that accept weights directly, such as naive Bayes and SVM.

As is the case for selection probabilities in Chapter VI, in practice we may not know the probabilities $P(y|x)$ used by the training policy in advance. However, we can estimate these from the available training data by using a classifier learning method coupled with the calibration methods presented in Chapter V.

Table VII.1 shows the pseudo-code for the one-step reinforcement learning with traces algorithm. Given a training set of the form (x, y, r), we first learn a model for $P(y|x)$. This can be accomplished by using a classifier learner that outputs calibrated class membership probability estimates. We then calculate weights for each example (x, y, r) by dividing r by $P(y|x)$. We can now use a

One-Step RL with traces(Training Set $S = (x, y, r)$)

1. **Learn a model for $P(y|x)$ using S.**
2. **Calculate a weight for each example (x, y, r): $w = \frac{r}{P(y|x)}$**
3. **Learn a classifier h using a cost-sensitive learner on $S' = (x, y, w)$.**
4. **Output h.**

Table VII.1: The one-step RL with traces algorithm.

cost-sensitive learning method that takes examples (x, y, w) as input, such as the ones presented in Chapter III to learn a classifier that is the desired the policy for the MDP.

VII.D T-step reinforcement learning with traces

Here, we introduce the trace model, which has the following operation:

- trace(π): returns a trace from the execution of a (non-deterministic) policy π on the MDP, that is, a sequence of state-action-reward triples of the form

$$(\langle s_0, a_0, r_1\rangle, \langle s_1, a_1, r_2\rangle, \ldots, \langle s_{T-1}, a_{T-1}, r_T\rangle),$$

where s_0 is drawn from $P(s_0)$, each a_i is drawn from $\pi(a|s_i)$, each s_i (for $0 < t < T$) is drawn from $P(s'|s_{i-1}, a_{i-1})$ and $r_i = R(s_i)$.

This model is weaker than a generative model since we can simulate the trace operation using a generative model operations but the opposite is not true.

We are going to reduce the problem of finding a good policy in this setting to T instances of classification, one for each timestep. The T learned classifiers are denoted c_t for $t \in \{1, \ldots, T\}$ and the nonstationary policy they create is $\pi(a|s, t) = c_t(s)$.

The optimization starts with some arbitrary initial policy, $c_1, \ldots, c_t$ and new classifiers c_t are learned iteratively, starting from $t = 1$ until $t = T$ and then going back to $t = 1$ and so forth.

For each optimization problem, we learn a new classifier c'_t, and then compare the two policies $c_1, \ldots, c'_t, \ldots, c_T$ and $c_1, \ldots, c_t, \ldots, c_T$ by using the trace model. If the new policy has a provably larger expected value, we replace $c_t \leftarrow c'_t$, then move on to the next optimization problem. The process halts after we observe T non-updates in a row.

The reduction for timestep t works by first calling the operation trace(π') m times, where the policy $\pi'(a|s,i)$ is given by $c_i(s)$ for $i < t$ and $i > t$. At timestep t, $\pi'(a|s,i)$ is an arbitrary distribution. Each returned trace is made into an example of the form (x, y, w), where $x = s_t$, $y = a_t$ and $w = \frac{1}{T}\sum_{t=i+1}^{T} r_t$.

Let

$$W(s_t, a_t) = \frac{1}{T}\sum_{i=t+1}^{T} R(s_i)$$

where $s_{t+1} \sim P(s'|s_t, a_t)$ and $s_{i+1} \sim P(s'|s_i, \pi'(s_i))$ for $t < i \leq T$.

Then $E[W(x, c_t(x))]$ is the expected value of the reward obtained by following $c_t(x)$ in step t while following π' in all the subsequent steps.

Theorem VII.C.1 shows that this expected value can be rewritten in a way that allows us to use a classifier learner that maximizes (VII.C.2) to learn the t-th step classifier c_t that maximizes $E[W(x, c_t(x))]$ using the examples (x, y, w).

The policy obtained from this reduction is the (approximately) best possible local improvement over the arbitrary policy used for training. "Local" here refers to the fact that we optimize the classifier c_t for each step t in a greedy manner, assuming that the policy will remain the same for the other steps. However, we continually change the policy to improve its overall value. The only guarantee is that each classifier c_t added will increase the overall policy value.

Even though this is a local reduction it is interesting because it is the first method for reinforcement learning in the policy mining setting, where we have traces of the execution of a (stochastic) training policy on the MDP. This set

of traces can be used to simulate the trace model by rejection sampling to produce a subset of traces consistent with the policy π given to the trace model.

VII.E Policy evaluation using a fixed dataset

The most obvious way to estimate the value of a policy given a fixed set of episodes is to select the episodes whose sequence of actions agrees with the policy. Then, we can average the cumulative rewards obtained in each of the selected episodes to obtain an estimate of the value of the policy. This is reasonable if the number of episodes in the set is large, the length of each episode is short and the number of possible actions is small, so that we can obtain enough episodes that agree with the policy.

Nonetheless, even when these conditions are true, selecting the available episodes in this manner will result in a biased estimate of the policy value. The initial state of the episodes used to evaluate the policy will not be an i.i.d. sample of the initial state distribution because the episodes are being selected according to a criteria that is not necessarily independent of the initial state. For example, in the direct marketing case, the initial state describes a particular customer. If we have a policy that is more likely to agree with the data for the "rich customers" (who presumably tend to donate more), by using this kind of evaluation we may think it is a very good policy. However, if we apply the policy to the general population of customers it may not perform as well.

In the policy evaluation case, we can translate the episodes into examples (x, y, s), such that x is an initial state, y is the cumulative reward obtained in the episode and s indicates whether the policy we would like to evaluate agrees with the actions in the episode. The expected value of y corresponds to the expected value of the policy, but we can only see the value of y when $s = 1$.

This is exactly the evaluation under sample selection bias problem presented in Chapter VI. Therefore, we can use the bias correction theorem to obtain

an unbiased estimate of the value of the policy. As stated before, this theorem assumes that y and s are conditionally independent given x. This assumption is reasonable when the selection is based solely on x. In the direct marketing case, the decision about mailing a customer depends only on x (the customer features) because y (the donation amount) is unknown. Therefore, if there is a dependence between y and s it disappears when we know x.

VII.F Applications

The methods presented in this chapter are useful for data mining applications in which the available data consists of records of an agent making decisions and receiving rewards. There is a surprisingly large number of practical data mining applications that can be cast in this framework, including direct marketing, fraud detection, recommender systems and medical treatment.

In all these cases, we are interested in discovering policies for how to act in different states of an environment. Depending on the domain, a state of the environment might describe a customer, a patient or a sequence of telephone calls. Because there are decision-making agents already in place for these applications (either humans or software systems), we can record the actions that the agents take and the rewards received by the agents.

Depending on the goals of the application, the nature of the rewards will vary. For example, in business applications (such as fraud detection), we are typically interested in maximizing profit, whereas in medicine one possible goal is to maximize the life span of the patient

The following subsections describe in more detail two of these applications for which there are datasets available publicly.

VII.F.1 Medical treatment

In medical treatment, we can think of the condition of a patient (which includes, for example, exam results) as a state. The doctor who prescribes a treatment is a decision-making agent and the different possible treatments are different actions. Depending on the treatment goals, we can use one of several measures to describe the success of a treatment program.

It is important to distinguish medical treatment from medical diagnosis. In medical diagnosis, we are interested in diagnosing a disease (for example, whether a patient has cancer or not). In medical treatment, we are interested in deciding what treatment to prescribe to a patient (for example, chemotherapy), based on evidence that the treatment is the best choice for the patient in the long run.

The problem of medical diagnosis can be easily cast as a classifier learning problem, where each example describes characteristics of a patient (such as exam results) and the label says whether or not the patient has the disease. In fact, the application of supervised learning techniques to medical diagnosis is fairly common. This is evidenced by the large number of medical datasets available in the UCI Machine Learning repository [10], such as Audiology, Breast Cancer, Dermatology, Diabetes, Hepatitis and Thyroid Disease.

On the other hand, medical treatment can only be cast as a classifier learning problem if we assume that the existing agent (i.e., the doctor) always takes the optimal actions. Then, we can use a classifier learning method to "capture" these actions and apply them to other patients. However, in general, we do not expect that the doctors will always take the optimal actions. In fact, the goal is to use the available data to learn what those actions should be for different patients based on how they react to the treatments. The policy mining presented in this chapter is the first that aims at solving this problem directly in a general manner.

A publicly available dataset that can be used for learning medical treatment policies is the Wisconsin Prognostic Breast Cancer Chemotherapy Dataset

(WPBCC) [92]. It contains medical information about 253 patients who have gone through breast cancer surgery (31 features). For each patient, there is a variable describing if the patient was given chemotherapy or not after surgery (140 of the patients were given chemotherapy) and the number of days lived after the surgery (survival time). The goal is to learn a policy based on the feature values for deciding which patients should be given chemotherapy, such that the expected number of days that each patient lives after the surgery is maximized.

Note that this particular medical treatment problem is not sequential because there is only one decision point: to give chemotherapy or not to a given patient described by the features. However, it is still challenging because chemotherapy is not given to a random set of patients. In fact, the average survival time for patients who were treated with chemotherapy (58.93) is less than the average survival time for patients who were not treated

The current approach for dealing with this data by Lee, Mangasarian and Wolberg [51] is to cluster the patients into three groups: Good, Intermediate and Poor. These groups strongly reflect patient survival times. None of the Good group patients receive chemotherapy and they have the highest average survival times. All of the Poor group patients receive chemotherapy and they have the lowest average survival times. About half of the patients in the Intermediate group receive chemotherapy, but those patients have better survival than the patients which did not receive it. Based on these results, they suggest that patients in the Good group should not receive chemotherapy and patients in the Intermediary group should receive therapy. The authors say that their approach is the first to identify a group of patients for which it is better to prescribe chemotherapy than not to prescribe it (the Intermediary group).

Note that the clustering results cannot be used on a new patient since it uses the chemotherapy feature (which is unavailable) and the lymph node status (which is very risky to obtain). For this reason, they learn a classifier for predicting Good, Intermediate or Poor for new patients based on available features. This

classifier achieves 82.7% accuracy on a test set.

A drawback of this clustering approach is that it is difficult to replicate it for other datasets. The clustering procedure "is based on physicians' knowledge and experience" and it involves a series of intermediary steps. Another problem is that the decision policy is very coarse-grained: it only separates the patients into three groups.

VII.F.2 Direct marketing

In direct marketing, an organization can encode the purchasing history of a customer and any other available data about the customer into a state. Any communication directed from the organization to the customer (such as a catalog mailing) is an action that changes the state of the customer and results in a reward to the organization (which may be positive or negative). Depending on the organization goals, different measures can be used to assess the success of its marketing strategy with respect to a customer. A common measure of success is the profit that the customer generates.

Many catalog mailing organizations have customer databases and software systems in place for deciding which customers should be mailed a catalog in a direct marketing campaign. Commonly, these systems are based on classifiers that try to distinguish customers who respond from customers who do not respond to a particular kind of catalog. Although these are likely not to be the optimal decisions with respect to profit in the long run, we can use the data collected in this manner to learn a mailing policy using policy mining.

A publicly available dataset that can be used for learning direct marketing policies is the dataset from the KDD-98 competition, described in Chapter II. Although the original task associated with this dataset concerns only the last campaign, it contains a detailed donation history of individuals who donated to the charity over a period of two years (22 campaigns). The data is divided into a training set and a test set, each containing 95412 and 96367 individuals, respec-

tively. For each campaign, we know whether each individual was mailed or not, whether he or she responded or not and how much was donated. Additionally, if the individual was mailed, the date of the mailing is available (month and year), and if the individual has responded, the date of the response is available.

The KDD-98 dataset has been used by Pednault et al. [62] to demonstrate the applicability of reinforcement learning to direct marketing problems. Based on the campaign information in the data, they generated a number of temporal features that are designed to capture the state of that individual at the time of each campaign. These include the frequency of gifts, recency of gift and promotion, number of recent promotions in the last 6 months, etc.

VII.G Experimental evaluation

VII.G.1 Quest Synthetic Data Generator

We first present experimental results using a synthetic data generator that is a modification of the IBM Quest Synthetic Data Generation Code for classification (Quest)[75]. Quest randomly generates examples for a person data set in which each person has the nine attributes described below.

- `Salary`: uniformly distributed between 20000 and 150000.
- `Commission`: if `Salary` $\geq$ 75000, `Commission` $= 0$, else uniformly distributed between 10000 and 75000.
- `Age`: uniformly chosen from 60 integer values (20 to 80).
- `Education`: uniformly chosen from 4 integer values.
- `CarMake`: uniformly chosen from 20 integer values.
- `ZipCode`: uniformly chosen from 9 integer values.
- `HouseValue`: uniformly distributed from $50000\,k$ to $150000\,k$, where $0 \leq k \leq 9$ and depends on the `ZipCode`.

- `YearsOwned`: uniformly distributed from 1 to 30.
- `Loan`: uniformly distributed between 0 and 500000.

In the original Quest generation code, there are a series of classification functions of increasing complexity that used the above attributes to classify people into different groups. After determining the values of different attributes of an example and assigning it a group label according to the classification function, the values for non-categorical attributes are perturbed. If the value of an attribute A for an example x is v and the range of values of A is a, then the value of A for x after perturbation becomes $v + r * a$, where r is a uniform random variable between -0.5 and +0.5.

We modified Quest to include both action generation functions and a reward generation functions. These are used in policy mining (instead of the classification functions) to generate examples of the form (x, y, r), where x is a person described by the attributes above, y is an action taken for that person (such as mailing a particular catalog) and r is the reward received after action y is taken (such as the amount purchased from the catalog).

The action generation function corresponds to a training policy. Given an example, it determines what action will be taken for that example. We use two different action generation functions that were created based on classification functions already implemented in Quest. These are shown in Table VII.2.

Given an example x and the action y taken for that example, the reward generation function determines a reward for executing action y with person x. We use two different reward generation functions, which are shown in Table VII.3.

The advantage of using a synthetic data generator is that we can evaluate any policy by generating the rewards for each possible action, which is not possible with real data. In the real-world, we cannot "reset" customers to the same state and mail a different catalog as if the customer had not received the first one, but we can do this with Quest.

Action Function 1

```
if (Age < 40)
  if (50000 ≤ Salary ≤ 100000)
    probAction1 = 0.3;
  else
    probAction1 = 0.7;
else
if (40 ≤ Age < 60)
  if (75000 ≤ Salary ≤ 125000)
    probAction1 = 0.1;
  else
    probAction1 = 0.9;
else
if (25000 ≤ Salary ≤ 75000)
    probAction1 = 0.4;
  else
    probAction1 = 0.6;

if (probAction1 > rand())
  action=1;
else
  action=0;
```

Action Function 2

```
if (Age < 40)
  probAction1 = 0.2;
else
if (40 ≤ Age < 60)
  probAction1 = 0.8;
else
  probAction1 = 0.2;

if (probAction1 > rand())
  action=1;
else
  action=0;
```

Table VII.2: Action generation functions. The function `rand()` generates a random number drawn uniformly from the interval $[0, 1]$.

Reward Function 1

```
if (YearsOwned < 20)
  equity = 0;
else
  equity = 0.1*YearsOwned - 2;

disposable = 2*Salary/3
             - Loan/5
             + 5000*Education
             + equity/5
             - 10000;

if (disposable > 0)
  if (action = 0)
    reward = randn(250,20);
  else
    reward = randn(200,20);
else
  if (action = 0)
    reward = randn(80,20);
  else
    reward = randn(150,20);
```

Reward Function 2

```
if (Age < 40)
  if (Education ∈ {0,1})
    if (action = 0)
      reward = randn(100,20);
    else
      reward = randn(80,20);
  else
    if (action = 0)
      reward = randn(50,20);
    else
      reward = randn(120,20);
else
if (40 ≤ Age < 60)
  if (Education ∈ {1,2,3})
    if (action = 0)
      reward = randn(100,20);
    else
      reward = randn(150,20);
  else
    if (action = 0)
      reward = randn(120,20);
    else
      reward = randn(140,20);
else
  if (Education ∈ {2,3,4})
    if (action = 0)
      reward = randn(90,20);
    else
      reward = randn(70,20);
  else
    if (action = 0)
      reward = randn(50,20);
    else
      reward = randn(70,20);
```

Table VII.3: Reward generation functions. The function randn(μ,σ) generates a random number drawn from a Gaussian with mean μ and standard deviation σ.

We applied the One-Step RL with traces algorithm VII.1 to three training sets of 50000 examples generated using three settings of the action and reward functions (Action1-Reward1, Action1-Reward2 and Action2-Reward2). For obtaining the estimates of $P(y|x)$ we use naive Bayes followed by the PAV calibration algorithm (see Chapter V). For learning the policy, we use three methods:

- weighted Naive Bayes,
- costing with Naive Bayes as base learner,
- costing with C4.5 as base learner.

For evaluating the policies, we use the simulator to generate three test sets of 50000 examples. We evaluate the policies using three methods:

- **True:** use the generator to obtain reward values for the two actions for each test example and average the rewards for the actions chosen by the policy (unbiased but unrealistic in a data mining setting).
- **Biased:** select only the test examples that agree with the policy and average the rewards for those examples.
- **Corrected:** select only the test examples that agree with the policy and use the bias correction method proposed in Section VII.E to calculate the expected reward of the policy (unbiased and realistic).

The probabilities $P(y|x)$ necessary for the bias correction method are obtained by applying the model learned on the training set to the test examples.

Table VII.4 summarizes the results obtained. For comparison purposes, it also includes the true expected value of the training policy, the best possible policy and the worst possible policy.

In all cases, the one-step RL with traces algorithm improves upon the training policy. Furthermore, by comparing the two settings with the same reward function and different training policies, we see that the training policy does not

Action Function 1 - Reward Function 1

	Evaluation Method		
Policy	True	Biased	Corrected
worst possible	146.94	-	-
best possible	206.31	-	-
training policy	178.06	-	-
weighted NB	192.74	180.74	191.86
costing NB	192.30	180.80	191.80
costing C.45	190.94	180.23	190.78

Action Function 1 - Reward Function 2

	Evaluation Method		
Policy	True	Biased	Corrected
worst possible	73.87	-	-
best possible	116.01	-	-
training policy	102.99	-	-
weighted NB	107.21	115.65	107.85
costing NB	107.06	115.53	107.76
costing C.45	112.45	120.30	112.56

Action Function 2 - Reward Function 2

	Evaluation Method		
Policy	True	Biased	Corrected
worst possible	73.87	-	-
best possible	116.01	-	-
training policy	96.12	-	-
weighted NB	107.08	112.20	108.50
costing NB	107.09	112.34	108.56
costing C.45	112.67	116.41	112.52

Table VII.4: Experimental results using Quest.

a great influence on the final result. The different learning algorithms (weighted NB, costing NB and costing C4.5) in general led to policies that are equally good, except that costing C4.5 resulted in a better policy for the settings with Reward2.

Whereas using only the selected examples to evaluate the policy yields incorrect estimates of the value of the policy, the evaluation using the bias correction method yields results that are very close to the true (and unrealistic) evaluation.

Bibliography

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[2] S. Anifantis. The DMEF data set library. The Direct Marketing Association, New York, NY, 2002. `http://www.the-dma.org/dmef/dmefdset.html`.

[3] M. Ayer, H. Brunk, G. Ewing, W. Reid, and E. Silverman. An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, 26(4):641–647, 1955.

[4] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139, 1999.

[5] J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning. Technical report, Australian National University, Research School of Information Sciences and Engineering, 1999.

[6] S. D. Bay. UCI KDD archive. Department of Information and Computer Sciences, University of California, Irvine, 2000. `http://kdd.ics.uci.edu/`.

[7] V. Bayer-Zubek. *Learning Cost-Sensitive Diagnostic Policies from Data.* PhD thesis, Department of Computer Science, Oregon State University, Corvallis, 2003.

[8] P. N. Bennett. Assessing the calibration of naive Bayes' posterior estimates. Technical Report CMU-CS-00-155, School of Computer Science, Carnegie Mellon University, 2000.

[9] C. Bishop. *Neural Networks for Pattern Recognition*, chapter 2. Clarendon Press, Oxford, UK, 1995.

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

[11] H. Bourland and N. Morgan. A continuous speech recognition system embedding mlp into hmm. In *Advances in Neural Information Processing Systems 2*, pages 186–193, 1990.

[12] J. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. Brodley. Pruning decision trees with misclassification costs. In *Proceedings of the European Conference on Machine Learning*, pages 131–136, 1998.

[13] U. Brefeld, P. Geibel, and F. Wysotzki. Support vector machines with example dependent costs. In *Proceedings of the Fourteenth European Conference on Machine Learning*, 2003.

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

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

[16] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.

[17] J. Cussens. Bayes and pseudo-Bayes estimates of conditional probabilities and their reliability. In *Proceedings of the European Conference on Machine Learning*, pages 136–152. Springer Verlag, 1993.

[18] M. H. DeGroot and S. E. Fienberg. The comparison and evaluation of forecasters. *Statistician*, 32(1):12–22, 1982.

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

[20] 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. ACM Press, 1999.

[21] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 105–112. Morgan Kaufmann Publishers, Inc., 1996.

[22] C. Drummond and R. C. Holte. Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 239–246, 2000.

[23] R. Duda, P. Hart, and D. Stork. *Pattern Classification.* John Wiley and Sons, 2001.

[24] L. Dümbgen. Statistical software (MATLAB), 2000. Available at `http://www.math.mu-luebeck.de/workers/duembgen/software/software.html`.

[25] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–261, 1989.

[26] C. Elkan. Boosting and naive bayesian learning. Technical Report CS97-557, University of California, San Diego, 1997.

[27] C. Elkan. Cost-sensitive learning and decision-making when costs are unknown. In *Workshop Notes, Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning*, 2000.

[28] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, Aug. 2001.

[29] C. Elkan. Magical thinking in data mining: Lessons from coil challenge 2000. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 426–431. ACM Press, 2001.

[30] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.

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

[32] W. Fan, H. Wang, P. Yu, and S. J. Stolfo. A framework for scalable cost-sensitive learning based on combining probabilities and benefits. In *Proceedings of the Second SIAM International Conference on Data Mining*, pages 97–105, 2002.

[33] 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):119–139, 1997.

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

[35] G. Fumera and F. Roli. Cost-sensitive learning in support vector machines. In *VIII Convegno Associazione Italiana per L'Intelligenza Artificiale*, 2002.

[36] P. Geibel and F. Wysotzki. Perceptron based learning with example dependent and noisy costs. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[37] J. Georges and A. H. Milley. KDD'99 competition: Knowledge discovery contest report. Available at `http://www-cse.ucsd.edu/users/elkan/kdresults.html`, 1999.

[38] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*, chapter 10.5. MIT Press, 2001.

[39] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998.

[40] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, chapter 2. Springer, 2001.

[41] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, chapter 4.4. Springer, 2001.

[42] D. Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[43] J. Heckman. Sample selection bias as a specification error. *Econometrica*, 47:153–161, 1979.

[44] T. Joachims. Estimating the generalization performance of a SVM efficiently. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 431–438, 2000.

[45] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 2000.

[46] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[47] M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17:115–141, 1998.

[48] U. Knoll, G. Nakhaeizadeh, and B. Tausend. Cost-sensitive pruning of decision trees. In *Proceedings of the Eight European Conference on Machine Learning*, pages 383–386, 1994.

[49] E. G. Kong and T. G. Dietterich. Probability estimation using error-correcting output coding. In *Int. Conf.: Artificial Intelligence and Soft Computing*, 1997.

[50] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[51] Y.-J. Lee, O. L. Mangasarian, and W. H. Wolberg. Survival-time classification of breast cancer patients. Technical Report 01-03, Data Mining Institute, University of Wisconsin, Madison, 2001.

[52] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[53] A. Likas and I. Lagaris. Training reinforcement neurocontrollers using the polytope algorithm. *Neural Processing Letters*, 9(2):119–127, 1999.

[54] R. Little and D. Rubin. *Statistical Analysis with Missing Data.* Wiley, 2nd edition, 2002.

[55] E. C. Malthouse. Assessing the performance of direct marketing scoring models. *Journal of Interactive Marketing*, 15(1):49–62, 2001.

[56] D. Margineantu. On class probability estimates and cost-sensitive evaluation of classifiers. In *Workshop Notes, Workshop on Cost-Sensitive Learning, International Conference on Machine Learning*, June 2000.

[57] D. Margineantu. *Methods for Cost-Sensitive Learning.* PhD thesis, Department of Computer Science, Oregon State University, Corvallis, 2001.

[58] D. Margineantu. Class probability estimation and cost-sensitive classification decisions. In *Proceedings of the Thirteenth European Conference on Machine Learning*, pages 270–281, 2002.

[59] A. Murphy and R. Winkler. Reliability of subjective probability forecasts of precipitation and temperature. *Applied Statistics*, 26(1):41–47, 1977.

[60] A. Y. Ng and M. Jordan. PEGASUS:A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[61] A. Y. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Neural Information Processing Systems 14*, 2002.

[62] E. Pednault, N. Abe, B. Zadrozny, H. Wang, W. Fan, and C. Apte. Sequential cost-sensitive decision making with reinforcement learning. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, pages 204–213, 2002.

[63] J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In *Advances in Large Margin Classifiers.* MIT Press, 1999.

[64] F. Provost and P. Domingos. Well-trained PETs: Improving probability estimation trees. CDER Working Paper #00-04-IS, Stern School of Business, New York University, NY, NY 10012, 2000.

[65] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.

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

[67] J. R. Quinlan. Boosting, bagging, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, 1996.

[68] J. Rennie and R. Rifkin. Improving multiclass text classification with the support vector machine. Technical Report AIM-2001-026.2001, MIT, 2001.

[69] R. Rifkin. SvmFu 3, 2001. Available at `http://five-percent-nation.mit.edu/SvmFu`.

[70] T. Robertson, F. Wright, and R. Dykstra. *Order Restricted Statistical Inference*, chapter 1. John Wiley & Sons, 1988.

[71] G. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Departement, 1994.

[72] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2002.

[73] P. Smyth, A. Gray, and U. Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 506–514. Morgan Kaufmann Publishers, Inc., 1995.

[74] J. R. Sobehart, R. M. Stein, V. Mikityanskaya, and L. Li. Moody's public firm risk model: A hybrid approach to modeling short term default risk. Technical report, Moody's Investors Service, Global Credit Research, 2000. Available at `http://www.moodysqra.com/ research/crm/53853.asp`.

[75] R. Srikant. IBM Quest Synthetic Data Generation Code, 1999. Available at `http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html`.

[76] R. Storn and K. Price. Differential evolution - a simple and effective adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[77] M. Strens and A. Moore. Direct policy search using paired statistical tests. In *Proceedings of the 18th International Conference on Machine Learning*, pages 545–552, 2001.

[78] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[79] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

[80] M. Szummer and T. Jaakkola. Information regularization with partially labeled data. In *Neural Information Processing Systems 15*, 2003.

[81] G. Tesauro. Temporal difference learning of backgammon strategy. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 451–457, 1992.

[82] J. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[83] K. Tumer and J. Ghosh. Theoretical foundations of linear and order statistics combiners for neural pattern classifiers. Technical Report TR-95-02-98, The Computer and Vision Research Center, The University of Texas at Austin, 1995.

[84] P. Turney. Types of cost in inductive concept learning. In *Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on Machine Learning*, 2000.

[85] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[86] V. Vapnik. *Statistical Learning Theory.* Wiley, Chichester, UK, 1998.

[87] J. von Neumann. Various techniques used in connection with random digits. *Applied Mathematics Series*, 12:36–38, 1951.

[88] M. G. Walker. *Probability Estimation for Classification Trees and DNA Sequence Analysis.* PhD thesis, Knowledge Systems Laboratory, Stanford University, 1992.

[89] X. Wang and T. G. Dietterich. Efficient value function approximation using regression trees. In J. Boyan, W. Buntine, and A. Jagota, editors, *Statistical Machine Learning for Large Scale Optimization, Neural Computing Surveys*, pages 51–54. 2000.

[90] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.

[91] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–282, 1992.

[92] W. Wolberg, Y. Lee, and O. Mangasarian. Winsconsin Prognostic Breast Cancer Chemotherapy Database. Computer Sciences Department, University of Wisconsin, Madison, 1999.
`[ftp://ftp.cs.wisc.edu/math-prog/cpo-dataset/machine-learn/cancer/WPBCC/]`.

[93] B. Zadrozny. Reducing multiclass to binary by coupling probability estimates. In *Advances in Neural Information Processing Systems 14*, pages 1041–1048, 2002.

[94] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 204–213. ACM Press, 2001.

[95] 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. Morgan Kaufmann Publishers, Inc., 2001.

[96] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, pages 694–699. ACM Press, 2002.

[97] B. Zadrozny, J. Langford, and N. Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the 2003 IEEE International Conference on Data Mining*, 2003. To appear.

[98] W. Zhang and T. G. Dietterich. Value function approximations and job-shop scheduling. In J. A. Boyan, A. W. Moore, and R. S. Sutton, editors, *Proceedings of the Workshop on Value Function Approximation.* Carnegie-Mellon University, School of Computer Science, 1995.