GETTING THE MOST OUT OF YOUR DATA: MULTITASK BAYESIAN NETWORK STRUCTURE LEARNING, PREDICTING GOOD PROBABILITIES AND ENSEMBLE SELECTION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Alexandru Niculescu-Mizil

August 2008

GETTING THE MOST OUT OF YOUR DATA: MULTITASK BAYESIAN NETWORK STRUCTURE LEARNING, PREDICTING GOOD PROBABILITIES AND ENSEMBLE SELECTION

Alexandru Niculescu-Mizil, Ph.D.

Cornell University 2008

First, I consider the problem of simultaneously learning the structures of multiple Bayesian networks from multiple related datasets. I present a multitask Bayes net structure learning algorithm that is able to learn more accurate network structures by transferring useful information between the datasets. The algorithm extends the score and search techniques used in traditional structure learning to the multitask case by defining a scoring function for *sets* of structures (one structure for each task) and an efficient procedure for searching for a high scoring set of structures. I also address the task selection problem in the context of multitask Bayes net structure learning. Unlike in other multitask learning scenarios, in the Bayes net structure learning setting there is a clear definition of task relatedness: two tasks are related if they have similar structures. This allows one to automatically select a set of related tasks to be used by multitask structure learning.

Second, I examine the relationship between the predictions made by different supervised learning algorithms and true posterior probabilities. I show that quasi-maximum margin methods such as boosted decision trees and SVMs push probability mass away from 0 and 1 yielding a characteristic sigmoid shaped distortion in the predicted probabilities. Naive Bayes pushes probabilities toward 0 and 1. Other models such as neural nets, logistic regression and bagged trees usually do not have these biases and predict well calibrated probabilities. I experiment

with two ways of correcting the biased probabilities predicted by some learning methods: Platt Scaling and Isotonic Regression. I qualitatively examine what distortions these calibration methods are suitable for and quantitatively examine how much data they need to be effective.

Third, I present a method for constructing ensembles from libraries of thousands of models. Model libraries are generated using different learning algorithms and parameter settings. Forward stepwise selection is used to add to the ensemble the models that maximize its performance. The main drawback of ensemble selection is that it builds models that are very large and slow at test time. This drawback, however, can be overcome with little or no loss in performance by using model compression.

BIOGRAPHICAL SKETCH

Alexandru Niculescu-Mizil is a Ph.D. candidate in the Computer Science department at Cornell University. He received a Masters of Science degree in Computer Science from Cornell University and a Magna Cum Laude Bachelors degree in Mathematics and Computer Science from University of Bucharest. His research interests are in machine learning and data mining. He conducted research in inductive transfer, graphical model structure learning, probability estimation, empirical evaluations, ensemble methods, and on-line learning. He was awarded a Distinguished Student Paper Award at the twenty second International Conference on Machine learning for the paper "Predicting Good Probabilities with Supervised Learning", and a Best Student Paper Award at the Conference on Learning Theory for the paper "Regret Bounds for Sleeping Experts and Bandits".

To Ema,

ACKNOWLEDGMENTS

I am grateful to have had Rich Caruana as my advisor. He taught me all I know about research. I thank him for the countless hours he spent side by side with me showing me how to run experiments, how to interpret results, how to write a paper, and so on. I thank him for all the guidance and advice he gave me over the years, and for teaching me not only how to do good research, but also how to be a good researcher. Many thanks also go to the rest of my committee, especially to Thorsten Joachims. I thank Lillian Lee for her advice and support, and Robert Kleinberg for encouraging me to engage in theoretical machine learning research.

I thank my office mates –Andre Allavena, Eric Breck, Steve Chong, Jeff Hartline, Filip Radlinski, and Matthew Schultz–, and the "guys next door" –Tom Finley, Art Munson, Daria Sorokina, and Benyah Shaparenko– for the countless academic and non-academic discussions, for putting up with me, and, above all, for being true friends. I thank Cindy Robinson for the fun chats we had over the years. Going to her office always brightened my day.

A great deal of gratitude goes to my family and friends in Romania. I thank my parents for the education they gave me over the years and for guiding me toward an academic career. I thank my best friend, Andrei Vlagali, who helped me keep my sanity on several occasions. My love goes to Alina, who came to live with me in Ithaca, even though this meant leaving behind her family and friends. Without her love and care I would have never made it. Last, but most important, I thank Ema, for the joy and excitement she brought to my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

OVERVIEW

The first part of this dissertation is concerned with Bayesian network structure learning. Bayesian networks are a standard tool for reasoning with uncertainty that encode compactly the probabilistic relationships between variables of interest. Bayes nets are specified by a directed acyclic graph (DAG), called the Bayes net structure, that encodes the statistical dependence and independence relationships between the variables, and a set of parametrized conditional probability functions. Learning the dependency structure from data provides invaluable information about the domain, making Bayesian networks a very powerful data analysis tool. For instance, learning a Bayes net from bird sighting data can help ecologists and ornithologists understand how environmental and human factors influence the abundance of different bird species. Or, learning a Bayes net from gene expression data can give microbiologists insights into the gene regulatory system.

My work is motivated by the observation that in many situations data is available for multiple related problems. Bird sighting data is available for different, ecologically distinct, regions in North America; gene expression data is available for multiple species. If the dependency structures of the related problems are similar, then useful information can be transfered among problems. Specifically, finding a direct statistical dependency (or lack thereof) between two variables in one problem provides additional evidence for the same relationship in the other problems. Chapter 2 presents a Bayesian network structure learning technique that is able to leverage this additional evidence in a principled manner. When compared to the traditional approach of learning the structures for each domain in isolation, without inter-problem transfer, my technique recovers significantly more accurate dependency structures, especially in situations where the data is scarce.

Part of this work has been presented in (Niculescu-Mizil & Caruana, 2007).

Another problem I address in this dissertation is that of predicting accurate class membership probabilities with supervised classification methods. The overwhelming majority of the work in supervised classification has focused on predicting the "correct" class for a given instance. In many cases, however, there is no "correct" class, but rather the instance has a certain probability of membership in each of the classes. Being able to accurately estimate these membership probabilities is required by many applications. This ability is key, for instance, when the predictions are used in a decision making process, when classifiers are used as parts of larger systems, or when dealing with varying misclassification costs.

In Chapter 3 I analyze the ability of predicting accurate class membership probabilities of several widely used supervised learning algorithms. The analysis shows that a number of learning algorithms, including boosted decision trees, support vector machines, and Naive Bayes, predict inaccurate class membership probabilities. This makes them unusable in applications where probability estimation is critical. To address this problem, I investigate techniques that "fix" the predictions of these learning algorithms transforming them into accurate class membership probability estimates. Parts of this chapter have been presented in (Niculescu-Mizil & Caruana, 2005b), and (Niculescu-Mizil & Caruana, 2005a).

The last chapter tackles the issue of obtaining high performing classifiers. Even small improvements in the performance of a classifier can lead to large gains if the classifier is widely used. Consider, for instance, credit card fraud detection, where a classifier would be used to distinguish fraudulent transactions from authentic ones. Since credit card companies process billions of transactions every year, even small improvements in the performance of the used classifier result in large gains from preventing fraudulent transactions and avoiding frustrating honest customers.

Ensembles of classifiers, *meta-classifiers* that combine the predictions of a set of *base classifiers* in order to obtain higher performance, have proved to be very effective at obtaining high performing classifiers. In Chapter 4 I present *Ensemble Selection*, an ensemble learning technique that yields some of the most powerful, high performing, general purpose classifiers to date. The work in this chapter has been presented in (Caruana et al., 2004), (Caruana et al., 2006) and (Bucila et al., 2006).

BIBLIOGRAPHY

Caruana, R., Munson, A., & Niculescu-Mizil, A. (2006). *Getting the most out of ensemble selection* (Technical Report 2006-2045). Cornell University. Full version of paper published at ICDM 2006.

Caruana, R., Niculescu-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble selection from libraries of models. *Proc. 21st International Conference on Machine Learning (ICML'04)*.

Fawcett, T., & Niculescu-Mizil, A. (2007). PAV and the ROC convex hull. *Machine Learning*, *68*, 97–106.

Niculescu-Mizil, A., & Caruana, R. (2005a). Obtaining calibrated probabilities from boosting. *Proc. 21st Conference on Uncertainty in Artificial Intelligence (UAI '05)*. AUAI Press.

Niculescu-Mizil, A., & Caruana, R. (2005b). Predicting good probabilities with supervised learning. *Proc. 22nd International Conference on Machine Learning (ICML'05)* (pp. 625–632).

Niculescu-Mizil, A., & Caruana, R. (2007). Inductive transfer for bayesian network structure learning. *Proc. 11th International Conf. on AI and Statistics*.

CHAPTER 2

INDUCTIVE TRANSFER FOR BAYESIAN NETWORK STRUCTURE LEARNING

2.1 Introduction

Bayes Nets (Pearl, 1988) provide a compact, intuitive description of the dependency structure of a domain by using a directed acyclic graph to encode probabilistic dependencies between variables. This intuitive encoding of the dependency structure makes Bayes Nets appealing in expert systems where expert knowledge can be encoded through hand-built dependency graphs. Acquiring expertise from humans, however, is difficult and expensive, so significant research has focused on learning Bayes Nets from data. The learned dependency graph also provides useful information about a problem and is often used as a data analysis tool. For example Friedman et al. (2000) used Bayes Nets learned from gene expression data to discover regulatory interactions between genes for a species of yeast.

Until now, Bayes Net structure learning research has focused on learning the dependency graph for one problem in isolation. In many situations, however, data is available for multiple related problems. In these cases, inductive transfer (Caruana, 1997; Baxter, 1997; Thrun, 1996) suggests that it may be possible to learn more accurate dependency graphs by *transferring* information between problems. For example, suppose that we want to learn the gene regulatory structure for a number of yeast species. Since the regulatory structures are very similar, learning that there is an interaction between two genes in one species of yeast should provide evidence for the existence of the same interaction in the other species.

In this chapter, we present an algorithm for learning the Bayes Net structures for multiple related tasks simultaneously. The method assumes that the true struc-

tures of the related tasks are similar. When this assumption is true, the presence or absence of arcs in some of the structures provides evidence for the presence or absence of those same arcs in the other structures. By taking into account such evidence the multitask structure learning algorithm we propose is able to learn more accurate network structures than its single-task structure learning counterpart.

We also tackle the task selection problem for multitask structure learning. Task selection is an important, but hard, problem in inductive transfer: given a set of tasks, select a subset to use as related tasks in a multitask learner. The main reason why automatic task selection is difficult in a general multitask learning setting is that there is no clear definition of task relatedness. Even if there exists an intuitive notion of task relatedness, it might be difficult to quantify. Furthermore the intuitive notion of relatedness might not correspond to a type of relatedness that the multitask learning algorithm is able to take advantage of. For example, playing tennis and running are, intuitively, related tasks, but if the particular multitask learning algorithm used can only transfer information about arm movements then multitask learning might not provide a benefit.

Unlike in the general case, in the multitask structure learning setting there exists a clear, quantifiable notion of task relatedness: two tasks are related if their structures are similar. Moreover, this is exactly the type of relatedness the multitask structure learning algorithm we study in this chapter takes advantage of. While computing task relatedness this way is not feasible since the true structures are unknown, it is possible to compute a good approximation using only the available data. We propose two such measures of task relatedness and a task selection algorithm for multitask structure learning. We show that, by selecting an appropriate set of tasks to use as related tasks, task selection further improves the performance of multitask structure learning.

The chapter starts with an overview of Bayes Net structure learning for a single problem, then describes the new multitask structure learning algorithm in Section 2.3. Section 2.3.4 provides an empirical evaluation of the new algorithm. Section 2.4 presents two task relatedness measures and the task selection algorithm. Empirical results supporting our task selection method are presented in Section 2.4.3. The chapter ends with an overview of related work in Section 2.5 and final conclusions in Section 2.6.

2.2 Learning Bayes Nets from Data

A Bayesian Network $\mathcal{B} = \{G, \theta\}$ that encodes the joint probability distribution of a set of n random variables $X = \{X_1, X_2, ..., X_n\}$ is specified by a directed acyclic graph (DAG) G and a set of conditional probability functions parametrized by θ (Pearl, 1988). The Bayes Net *structure*, G, encodes the probabilistic dependencies in the data: the presence of an edge between two variables means that there exists a direct dependency between them. An appealing feature of Bayes Nets is that the dependency graph G is easy to interpret and can be used to aid understanding the problem domain.

Given a dataset $D = \{x^1, ..., x^m\}$ where each x^i is a complete assignment of variables $X_1, ..., X_n$, it is possible to learn both the structure G and the parameters θ (Cooper & Hersovits, 1992; Heckerman, 1999). Following the Bayesian paradigm, the posterior probability of the structure given the data is estimated via Bayes rule:

$$P(G|D) \propto P(G)P(D|G) \tag{2.1}$$

The prior $P(G)$ indicates the belief before seeing any data that the structure G is correct. If there is no reason to prefer one structure over another, one should assign the same probability to all structures. This uninformative (uniform) prior is

rarely accurate, but often is used for convenience. If there exists a known ordering on the nodes in G such that all the parents of a node precede it in the ordering, a prior can be assessed by specifying the probability that each of the $n(n-1)/2$ possible arcs is present in the correct structure (Buntine, 1991). Alternately, when there is access to a structure believed to be close to the correct one (e.g. from an expert), $P(G)$ can be specified by penalizing each difference between G and the given structure by a constant factor (Heckerman et al., 1995).

The marginal likelihood, $P(D|G)$, is computed by integrating over all possible parameter values:

$$P(D|G) = \int P(D|G,\theta)P(\theta|G)d\theta \tag{2.2}$$

When the local conditional probability distributions are from the exponential family, the parameters θ_i are mutually independent, we have conjugate priors for these parameters, and the data is complete, $P(D|G)$ can be computed in closed form (Heckerman, 1999).

Treating $P(G|D)$ as a score, one can search for a high scoring network using heuristic search (Heckerman, 1999). Greedy search, for example, starts from an initial structure, evaluates the score of all the *neighbors* of that structure and moves to the neighbor with the highest score. The search terminates when the current structure is better than all it's neighbors. Because it is possible to get stuck in a local minima, this procedure usually is repeated a number of times starting from different initial structures. A common definition of the *neighbors* of a structure G is the set of all the DAGs that can be obtained by removing or reversing an existing arc in G, or by adding an arc that is not present in G.

2.3 Learning from Multiple Related Tasks

In the previous section we reviewed how to learn a Bayes Net for a single task. What if instead of a single task we have a number of related tasks (e.g., gene expression data for a number of related species) and we want to learn a Bayes Net structure for each of them?

Given k data-sets, $D_1, ..., D_k$, defined on overlapping but not necessarily identical sets of variables, we want to learn the structures of the Bayes Nets $\mathcal{B}_1 = \{G_1, \theta_1\}$, ..., $\mathcal{B}_k = \{G_k, \theta_k\}$, one for each data-set. In what follows, we will use the term *configuration* to refer to a set of structures $(G_1, ..., G_k)$.

From Bayes rule, the posterior probability of a configuration given the data is:

$$P(G_1, ..., G_k | D_1, ..., D_k) \propto P(G_1, ..., G_k) P(D_1, ..., D_k | G_1, ..., G_k) \tag{2.3}$$

The marginal likelihood $P(D_1, ..., D_k | G_1, ..., G_k)$ is computed by integrating over all parameter values for all the k networks:

$$\begin{aligned} P(D_1, ..., D_k | G_1, ..., G_k) &= \\ &= \int P(D_1, ..., D_k | G_1, ..., G_k, \theta_1, ..., \theta_k) \times \\ &\qquad P(\theta_1, ..., \theta_k | G_1, ..., G_k) d\theta_1 ... d\theta_k \\ &= \int P(\theta_1, ..., \theta_k | G_1, ..., G_k) \prod_{p=1}^{k} P(D_p | G_p, \theta_p) d\theta_1 ... d\theta_k \end{aligned} \tag{2.4}$$

If we make the parameters of different networks independent *a priori* (i.e. $P(\theta_1, ..., \theta_k | G_1, ..., G_k) = P(\theta_1 | G_1) ... P(\theta_k | G_k)$), the marginal likelihood becomes just the product of the marginal likelihoods of each data set given its network structure. In this case the posterior can be written as:

$$P(G_1, .., G_k | D_1, .., D_k) \propto P(G_1, .., G_k) \prod_{p=1}^{k} P(D_p | G_p) \tag{2.5}$$

Making the parameters independent *a priori* is unfortunate, and contradicts the intuition that related tasks should have related parameters, but it is needed in order to make structure learning efficient (see Section 2.3.3). It is important to note that this is not a restriction on the model. Unlike the Naive Bayes model for example, where the attribute independence assumption actually restricts the class of models that can be learned, here the learned parameters will be correlated if such correlation is present in the data. The only downside of making the parameters independent *a priori* is that it prevents multitask structure learning from taking advantage of the similarities between the parameters of different tasks during the structure learning phase. After the structures have been learned, however, such similarities could be leveraged to learn more accurate parameters. Finding ways to allow for some *a priori* parameter dependence while still maintaining computational efficiency is an interesting direction for future work.

2.3.1 The Prior

The prior knowledge of how related the different tasks are and how similar their structures should be is encoded in the prior $P(G_1, ..., G_k)$. If there is no reason to believe that the structures for each task should be related, then $G_1, ..., G_k$ should be made independent *a priori* (i.e. $P(G_1, ..., G_k) = P(G_1) \cdot ... \cdot P(G_k)$). In this case the structure-learning can be done independently for each task using the corresponding data set.

At the other extreme, if the structures for all the different tasks should be identical, the prior $P(G_1, ..., G_k)$ should put zero probability on any configuration that contains nonidentical structures. In this case one can efficiently learn the same structure for all tasks by creating a new data set with attributes $X_1, ..., X_n, TSK$, where TSK encodes the task the case is coming from.[1] Then learn the structure

[1]This is different from pooling the data, which would mean that not only the structures, but

for this new data set under the restriction that TSK is always the parent of all the other nodes. The common structure for all the tasks is exactly the learned structure, with the node TSK and all the arcs connected to it removed. This approach, however, does not easily generalize to the case where tasks have only partial overlap in their attributes. The algorithm proposed below avoids this problem, while computing the same solution when structures are forced to be identical.

Between these two extremes, the prior should encourage finding similar network structures for the tasks. The prior can be seen as penalizing structures that deviate from each other, so that deviation will occur only if it is supported by enough evidence in the data.

One way to generate such a prior for two structures is to penalize each arc (X_i, X_j) that is present in one structure but not in the other by a constant $\delta \in [0, 1]$:

$$P(G_1, G_2) = Z_\delta \cdot (P(G_1)P(G_2))^{\frac{1}{1+\delta}} \prod_{\substack{(X_i,X_j)\in \\ G_1 \Delta G_2}} (1-\delta) \tag{2.6}$$

where Z_δ is a normalization factor that is absorbed in the proportionality constant of equation 2.5, and $G_1 \Delta G_2$ represents the symmetric difference between the edge sets of the two DAGs (arc reversals can be counted only once or twice).

If $\delta = 0$ then $P(G_1, G_2) = P(G_1)P(G_2)$, so the structures are learned independently. If $\delta = 1$ then $P(G_1, G_2) = \sqrt{P(G)P(G)} = P(G)$ for $G_1 = G_2 = G$ and $P(G_1, G_2) = 0$ for $G_1 \neq G_2$, leading to learning identical structures for all tasks. For δ between 0 and 1, the higher the penalty, the higher the probability of more similar structures. The advantage of this prior is that $P(G_1)$ and $P(G_2)$ can be any structure priors that are appropriate for the task at hand. If a variable, X_i, is present in one structure but not in the other, then any arc that has X_i as one of its extremities should not incur any penalty.

also the parameters for all tasks will be identical.

One way to interpret the above prior is that it penalizes by δ each *edit* (i.e. arc addition, arc removal or arc reversal) that is necessary to make the two structures identical (arc reversals can count as one or two edits). This leads to a natural extension to more than two tasks that penalizes each edit that is necessary to obtain a set of identical structures:

$$P(G_1, ..., G_k) = Z_{\delta,k} \cdot \prod_{1 \leq s \leq k} P(G_s)^{\frac{1}{1+(k-1)\delta}} \times \prod_{i,j} (1-\delta)^{edits_{i,j}} \tag{2.7}$$

where $edits_{i,j}$ is the minimum number of edits necessary to make the edge between X_i and X_j the same in all the structures. We will call this prior the *Edit* prior. The exponent $1/(1+(k-1)\delta)$ is used to transition smoothly between the case where structures should be independent (i.e. $P(G_1, ..., G_k) = (P(G_1)...P(G_k))^1$ for $\delta = 0$) and the case where structures should be identical (i.e. $P(G, .., G) = (P(G)...P(G))^{1/k}$ for $\delta = 1$). This prior can be easily generalized by using different penalties for different edges (e.g. if certain edges should not chance between tasks then the penalty on those edged should be 1), and/or different penalties for different edit operations.

Another way to specify a prior for more than two tasks is to multiply the penalties incurred between all pairs of structures:

$$P(G_1, ..., G_k) = Z_{\delta,k} \cdot \prod_{1 \leq s \leq k} P(G_s)^{\frac{1}{1+(k-1)\delta}} \times \prod_{1 \leq s < t \leq k} \left(\prod_{\substack{(X_i, X_j) \in \\ G_s \Delta G_t}} (1-\delta) \right)^{\frac{1}{k-1}} \tag{2.8}$$

We will call this prior the *Paired* prior. The exponent $1/(k-1)$ is used because each individual structure is involved in $k-1$ terms (one for each other structure).

One advantage that the Paired prior has over the Edit prior is that it can be generalized by specifying different penalties between different pairs of structures. This can handle situations where there is reason to believe that Task1 is related to

Task2, and Task2 is related to Task3, but the relationship to between Task1 and Task3 is weaker.

There are of course other priors that encourage finding similar networks for each task in different ways. In particular, if the process that generated the related tasks is know, it might be possible to design a suitable prior.

2.3.2 Greedy Structure Learning

Treating $P(G_1, ..., G_k|D_1, ..., D_k)$ as a score, we can search for a high scoring configuration using an heuristic search algorithm. If we choose to use greedy search for example, we start from an initial configuration, compute the scores of the neighboring configurations, then move to the configuration that has the highest score. The search ends when no neighboring configuration has a higher score than the current one.

One question remains: what do we mean by the neighborhood of a configuration? An intuitive definition of a neighbor is the configuration obtained by modifying a single arc in a single DAG in the configuration, such that the resulting graph is still a DAG. With this definition, the size of the neighborhood of a configuration is $O(k * n^2)$ for k tasks and n variables. Unfortunately, this definition creates a lot of local minima in the search space. Consider for example the case where there is a strong belief that the structures should be similar (i.e. the penalty parameter of the prior, δ, is near one resulting in a prior probability near zero when the structures in the configuration differ). In this case it would be difficult to take any steps in the greedy search since modifying a single edge for a single DAG would make it different from the other DAGs, resulting in a very low posterior probability (score).

To correct this problem, we define the neighborhood of a configuration to be

the set of all configurations obtained by selecting two nodes, and for each structure in the configuration, add, remove, reverse, or leave unchanged the arc between the two selected nodes, under the restriction that the resulting structure remains a DAG. It is easy to see that there is a path between any two configurations, so the search space is connected. Given this definition, the size of a neighborhood is $O(n^2 3^k)$, which is exponential in the number of tasks, but only quadratic in the number of nodes.[2] In the case where all the learned structures are required to be identical (infinite penalty for diverging structures) multitask learning, with this definition of neighborhood, will find the same structures as the specialized algorithm described in Section 2.3.1. We will use this definition for the rest of the chapter.

2.3.3 Searching for the Best Configuration

At each iteration, the greedy procedure described in the previous section must find the best scoring configuration from a set $\mathcal{N}$ of neighboring configurations. In the naive approach the score of every configuration in $\mathcal{N}$ is computed and the configuration with the highest score is selected. Since the size of $\mathcal{N}$ can get large for large n or k, this naive approach can be very expensive. Much of this computation however can be avoided by using better search techniques to find the best scoring configuration.

Let a partial configuration of order l, $\mathcal{C}_l = (G_1, .., G_l)$, be a configuration where only the structures for the *first* l tasks are specified and the rest of $k - l$ structures are not specified. We say that a configuration $\mathcal{C}$ matches a partial configuration $\mathcal{C}_l$ if the structures for the first l tasks in $\mathcal{C}$ are the same as the structures in $\mathcal{C}_l$.

[2] The restriction that changes, if any, have to occur between the same nodes in all the structures could be dropped, but this would lead to a neighborhood that is exponential in both n and k. Considering the assumption that the structures should be similar, such a restriction is not inappropriate.

A search strategy for finding the best scoring configuration in $\mathcal{N}$ can be represented via a search tree of depth k that satisfies the following properties: a) each node at level l contains a different valid partial configuration of order l; b) all nodes in the subtree rooted at node $\mathcal{C}_l$ contain only (partial) configurations of order at least $l+1$ that match $\mathcal{C}_l$. (i.e. the first l structures are the same as in $\mathcal{C}_l$.)

If, given a partial configuration, the score of any complete configuration that matches it can be efficiently upper bounded, and the upper bound is lower than the current best score, then the entire subtree rooted at the respective partial configuration that can be pruned. This suggests using a branch and bound procedure for finding the best scoring configuration in $\mathcal{N}$, by using deep first search and pruning the current subtree whenever possible. This branch and bound search significantly reduces the number of partial configurations (and consequently complete configurations) that need to be explored.

Let $edits_{l,i,j}$ be the minimum number of edits necessary to make the edge between X_i and X_j the same in the *first* l structures, and let

$$Best_q = \max\{P(G_q)^{\frac{1}{1+(k-1)\delta}} P(D_q|G_q)\}.$$

If the marginal likelihood of a configuration factorizes in the product of the marginal likelihoods of the individual structures, as in equation 2.5, then the score of any configuration that matches the partial configuration $\mathcal{C}_l = (G_1, ..., G_l)$ can be upper bounded by:

$$U_{\mathcal{N}}^{Edit}(\mathcal{C}_l) = Z_{\delta,k} \cdot \left(\prod_{i,j}(1-\delta)^{edits_{l,i,j}}\right) \times \tag{2.9}$$
$$\times \quad \left(\prod_{1\leq p\leq l} P(G_p)^{\frac{1}{1+(k-1)\delta}} P(D_p|G_p)\right) \cdot \left(\prod_{l+1\leq p\leq k} Best_q\right)$$

if using the Edit prior (equation 2.7), and by

$$U_{\mathcal{N}}^{Paired}(\mathcal{C}_l) = Z_{\delta,k} \cdot \left(\prod_{1 \le s < t \le l} \prod_{\substack{(X_i, X_j) \in \\ G_s \Delta G_t}} (1-\delta) \right)^{\frac{1}{k-1}} \times \tag{2.10}$$

$$\times \left(\prod_{1 \le p \le l} P(G_p)^{\frac{1}{1+(k-1)\delta}} P(D_p|G_p) \right) \cdot \left(\prod_{l+1 \le p \le k} Best_q \right)$$

if using the Paired prior (equation 2.8).[3]

Note that for both these upper bounds, the fact that the marginal likelihood a configuration factorizes into the product of the marginal likelihoods of the individual structures plays a critical role. It allows us to both compute the exact contribution made by the specified structures to the marginal likelihood and to easily compute the maximum contribution the unspecified structures can make to the marginal likelihood of a configuration.

Another source of computational savings is the precomputation of the individual marginal likelihoods. With the definition of a neighborhood we are using, a neighboring configuration will have, in each of the k components, one of the $2n^2$ or fewer individual DAGs that differ by exactly one edge from the current DAG in the respective component. Each of these $2n^2$ (or fewer) DAGs are present in about 3^{k-1} neighboring configurations. Since a configuration score has the form in equation 2.5, the marginal likelihoods for the individual DAGs, $P(D_i|G_i)$, can be reused, thus reducing by a factor of about 3^{k-1} the expense of computing the marginal likelihoods of the neighboring configurations. It is also worth mentioning that both the prior and the likelihood are decomposable, so evaluating the score of the neighboring configurations requires only local computations.

[3]For the Paired prior it is possible to get a tighter upper bound, but we will use this one for simplicity.

2.3.4 Empirical Evaluation

We evaluate the performance of multitask structure learning using multitask problems generated by perturbing the ALARM (Beinlich et al., 1989) and INSURANCE (Binder et al., 1997) networks. We also evaluate the multitask structure learning algorithm on a real problem in bird ecology.

Data Sets

For the experiments with the ALARM and INSURANCE networks, we generate multiple related tasks by perturbing the original structures. We use two qualitatively different methods for perturbing the networks: randomly deleting edges, and changing entire subgraphs.

For the first method, for each problem, we create five related tasks by starting with the original network and deleting arcs with probability P_{del}. This way, the structures of the five tasks can be made more or less similar by varying P_{del} (For $P_{del} = 0$ all structures are identical).

Given the restriction we imposed in Section 2.3 that parameters for different tasks should be independent *a priori*, we want to investigate the performance of multitask structure learning in settings where the parameters are indeed independent between tasks, as well as in settings where the parameters are actually correlated between tasks. To this end, we create four multitask learning problems; two where the parameters are independent between tasks, and two where parameters are correlated between tasks. For the two problems with correlated parameters, denoted ALARM and INSURANCE, we start with the original structures and parameters, and perturb the structures as described above. When an arc is deleted, the parameters of the network are recomputed by integrating over the deleted parent, so that the dependency between the child and the remaining

parents is unchanged. This yields five related tasks with correlated parameters. To generate the two problems with independent parameters between tasks, denoted ALARM-IND and INSURANCE-IND, we also start with the original structures, but for each task, we use random parameters instead of the original ones. Then, we again perturb the structures and integrate over the deleted parent when an arc is removed.

We also experiment with a qualitatively different way of generating related tasks, ALARM-COMP. We split the ALARM network in 4 components: nodes 1-7 in the first component, nodes 9-14, 21 and 34 in the second, nodes 8,27-31, 36 and 37 in the third and the rest in the fourth component. For each of the five tasks, we randomly change the structure and parameters of zero, one or two of the components, while keeping the rest of the Bayes net (including parameters) unchanged. The first tasks consists of the original ALARM network, the second task has the first component changed, the third task has the second component changed, the fourth task has the third component changed and the fifth task has both the first and the third components changed. This way parts of the structures are shared between tasks while other parts are completely unrelated (see Figure 2.6). This method of creating related tasks tries to simulate the situation where whole pieces of the gene regulatory structures differ from one organism to another.

We also evaluate the performance of multitask structure learning on a real bird ecology problem. The data for this problem comes from Project FeederWatch (PFW, `http://birds.cornell.edu/pfw`), a winter-long survey of North American birds observed at bird feeders. Each PFW location and submission is described by multiple attributes. These attributes can be roughly grouped into features related to observer effort, weather during the observation period, and attractiveness

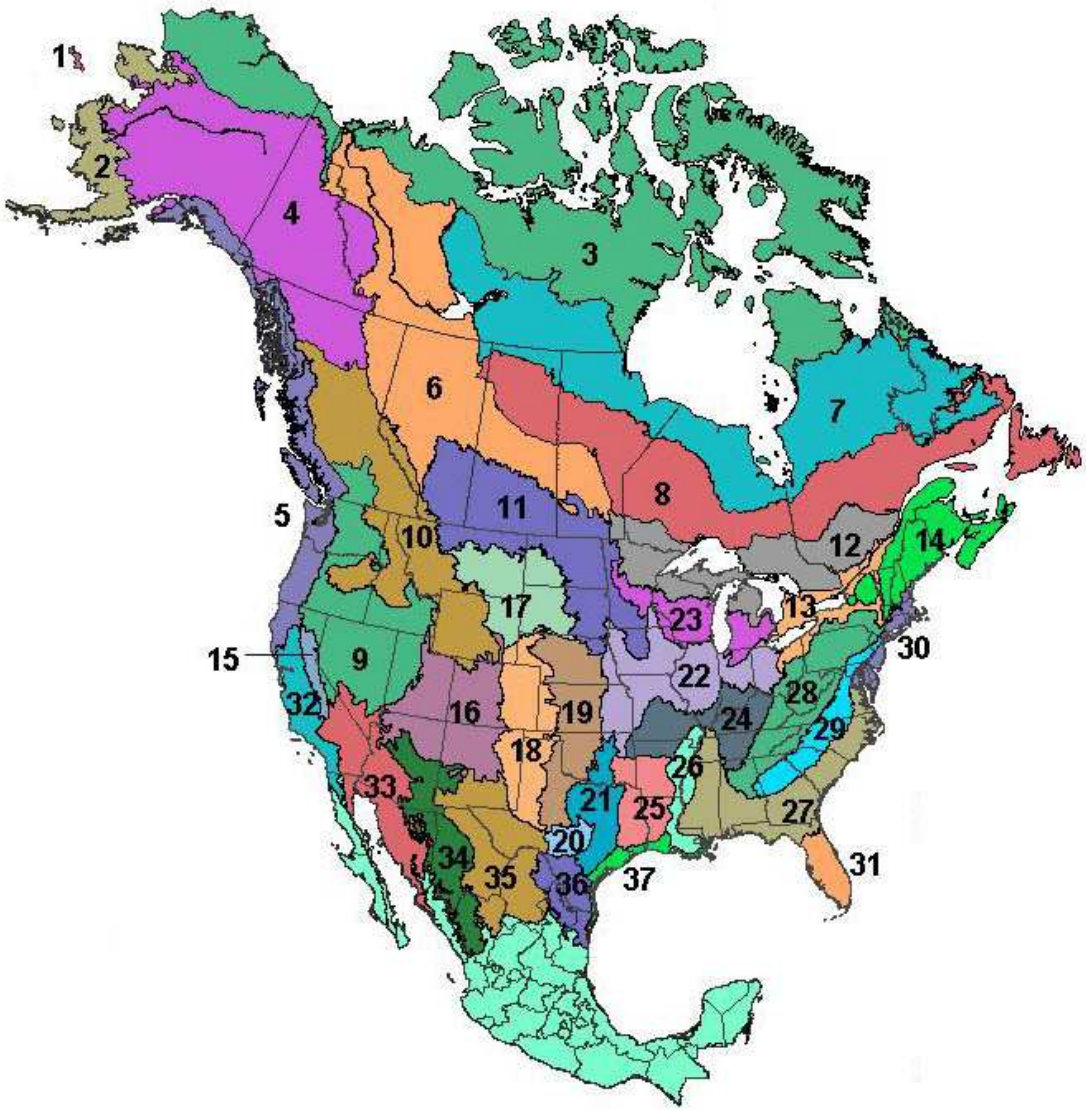


Figure 2.1: North American Bird Conservation Regions.

of the location and neighborhood area for birds. In this chapter we only examine the case where the data is fully observed, so we preprocess the PFW data by eliminating attributes that contain a large number of missing values, and by eliminating instances that still contain missing values in the remaining attributes.

Ecologists have divided North America into a number of ecologically distinct Bird Conservation Regions (BCRs; see Figure 2.1). This division naturally splits

the data into multiple tasks, one task per BCR. Because each bird species lives in some BCRs but not in others, and because there is a variable for each bird species in a BCR, this is an instance of a problem where the different tasks are not defined over identical sets of variables.

Although multitask structure learning is most beneficial for BCRs that have only a small amount of data, small amounts of data make evaluation difficult. In order to have multiple trials that are not too similar, and to have large enough test sets to ensure accurate estimates of generalization performance, in this chapter we focus on the BCRs with larger amounts of data. In this section we will use six BCRs as related tasks: 30, 29, 28, 22, 13 and 23. We justify the choice of these particular BCRs in Section 2.4.3.

Methodology

We compare multitask structure learning to single-task structure learning, and learning identical structures for all tasks. Single-task structure learning uses greedy hill-climbing with 100 restarts and tabu lists to learn the structure of each task independently of the others. The learning of identical structures is performed via the algorithm presented in Section 2.3.1 and it also uses greedy hillclimbing with 100 restarts and tabu lists.[4]

multitask structure learning uses the greedy algorithm from in Section 2.3.2 with the solution found by single-task learning as the starting point.[5] When needed, the penalty parameter of the multitask prior, δ, is selected using the following simple wrapper method:

[4]Learning identical structures and single-task structure learning can be viewed as learning an augmented naive Bayesian network and a Bayesian multi-net (Friedman et al., 1997) respectively, where the "class" of each example is the task it belongs to . Unlike in the usual setting, however, here we are not interested in predicting to which task an example belongs to. We are only interested in recovering accurate network structures for each task.

[5]Initializing MTL search with the STL solution does not provide an advantage to MTL, but makes the search more efficient.

1. Split the available training data into a training set and a small validation set.
2. Run the multitask structure learning algorithm on the training set with different values for the penalty parameter.
3. Select the value of the penalty parameter that yields the highest mean log likelihood on the small validation set.
4. Once a penalty parameter is selected and the structures have been learned, use both the training and validation sets to learn the Bayes Net parameters.

Note that for single task learning and learning identical tasks, where there are no free parameters, all available training data, including the data multitask learning uses as a validation set, is used to learn both structures and parameters in order to keep the comparison fair. For all methods, the Bayes net parameters are learned using Bayesian updating (see e.g. (Cooper & Hersovits, 1992)).

The goal is to recover as closely as possible the true Bayes Net structures for all the related tasks. The main measure of performance we use is average edit distance[6] between the true structures and learned structures. Edit distance directly measures the quality of the learned structures, independently of the parameters of the Bayes Net. We also measure the average empirical KL-divergence (computed on a large test set) between the distributions encoded by the true networks and the learned ones. Since KL-Divergence is also sensitive to the parameters of the Bayes Net it does not measure directly the quality of the learned structures, but, in general, more accurate structures lead to models with lower KL-Divergence. For the bird ecology problem, where the true networks are unknown, we measure performance in terms of mean log likelihood on a large independent test set.

[6]Edit distance measures how many edits (arc additions, deletions or reversals) are needed to get from one structure to the other.

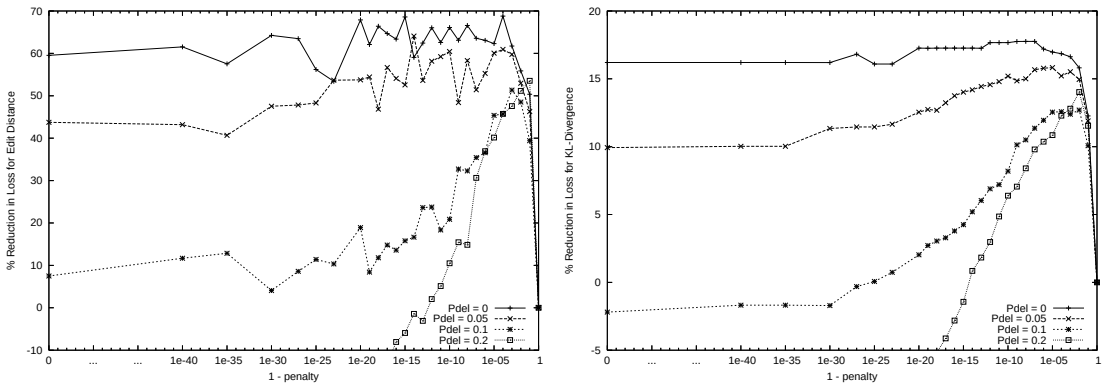


Figure 2.2: Reduction in edit distance (left) and KL-Divergence (right) for ALARM

The ALARM and INSURANCE problems

Figures 2.2 and 2.3 show the average percent reduction in loss, in terms of edit distance and KL-divergence, achieved by multitask learning over single-task learning for a training set of 1000 points on the ALARM and INSURANCE-IND problems. The figures for the ALARM-IND and INSURANCE problems are similar and are not included. On the x-axis we vary the penalty parameter of the multitask prior on a log-scale.[7] Note that the x-axis plots $1 - penalty$. The higher the penalty (the lower $1 - penalty$), the more similar the learned structures will be, with all the structures being identical for a penalty of one ($1 - penalty = 0$, left end of graphs). Each line in the figure corresponds to a particular value of P_{del}. Error bars are omitted to maintain the figure readable.

The trends in the graphs are exactly as expected. For all values of P_{del}, as the penalty increases, the performance increases because the learning algorithm takes into account information from the other tasks when deciding whether to add a new arc or not. If the penalty is too high, however, the algorithm loses the ability to find true differences between tasks and the performance drops. As the tasks become more similar (lower values of P_{del}), the best performance is obtained at

[7]The log-scale is needed because we are working in the probability space so $1 - \delta$ needs to change by orders of magnitude for the effects to be noticeable.

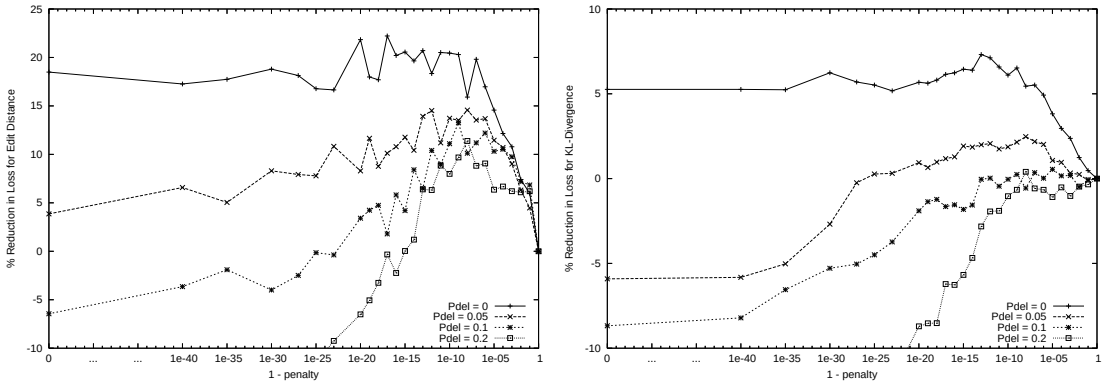


Figure 2.3: Reduction in edit distance (left) and KL-Divergence (right) for INSURANCE-IND

higher penalties. Also as the tasks become more similar, more information can be extracted from the related tasks, so usually multitask learning provides more benefit. As expected, multitask structure learning provides a larger improvement in edit distance than in KL-divergence. This happens because multitask structure learning helps to correctly identify the arcs that encode weaker dependencies (or independences) which have a smaller effect on KL-divergence. The arcs that encode strong dependencies, and have the biggest effect on KL-divergence, can be easily learned without help from the other tasks. multitask learning provides similar benefits whether the tasks have highly correlated parameters (ALARM and INSURANCE problems) or independent parameters (ALARM-IND and INSURANCE-IND problems). This shows that making the parameters independent *a priori* (see Section 2.3) does not hurt the performance of multitask learning. However, if we were able to take advantage of the similarity between the parameters of the different tasks, we could presumably improve performance even further.

Figure 2.4 shows the edit distance (left) and KL-divergence (right) performance of multitask structure learning when using the different multitask priors proposed in Section 2.3.1: the Paired prior from equation 2.8 with the reversed edges penalized twice (Paired/Double) or only once (Paired/Single) and the Edit prior from

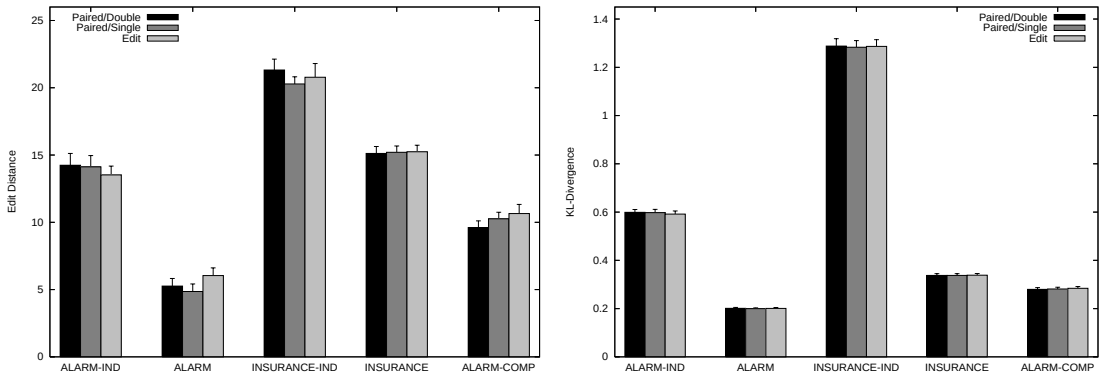


Figure 2.4: Edit distance (left) and KL-Div (right) for different multitask priors

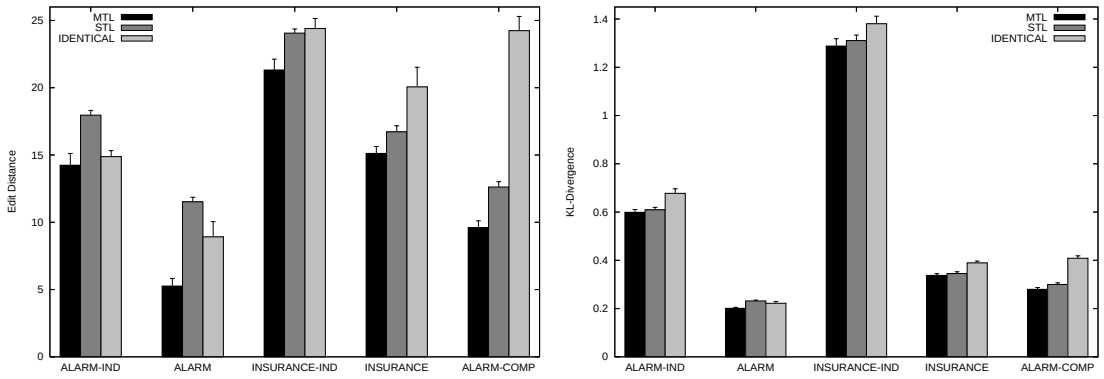


Figure 2.5: Edit distance (left) and KL-Div (right) for STL, learning identical structures and MTL

equation 2.7. Each group of bars corresponds to one problem. For ALARM-IND, ALARM, INSURANCE-IND and INSURANCE P_{del} is set to 0.05. The training set has 1000 cases, with a validation set of 50 cases for selecting the penalty parameter for the multitask prior as described in the beginning of this section. While there is some variability between the performance of the different priors, it is quite small, and never statistically significant. This suggests that multitask learning is relatively robust to the specific type of multitask prior, as long as it appropriately encourages sharing between the tasks. So one can safely use either type of prior without having to worry about selecting the best prior for the problem. Unless otherwise specified, in his chapter we use the Paired prior with double penalty on reversed edges.

Figure 2.5 shows the edit distance and KL-Divergence performance for single task learning (STL), learning identical networks via the algorithm presented in Section 2.3.1 (IDENTICAL), and multitask learning (MTL) for the five problems. The training set has 1000 instances with 50 instances used to select the penalty parameter for the multitask prior. Single-task learning and identical structure learning use both the training and the validation data to learn both the structure and the parameters of the Bayes Nets. The figure shows that multitask learning yields a 10%-54% reduction in edit distance and a 2% - 13% reduction in KL-divergence when compared to single task structure learning. All differences except for KL-divergence on ALARM-IND and INSURANCE-IND problems are .95 significant according to paired T-tests. When compared to learning identical structures, multitask learning reduces the KL-divergence 7% - 32% and the number of incorrect arcs in the learned structures by 4% - 60%. All differences are .95 significant, except for edit distance on the ALARM-IND problem. Since the five tasks for the ALARM, INSURANCE, and ALARM-COMP problems share a large number of their parameters, simply pooling the data might work well. However, this is not the case. Except for the ALARM problem, where it achieves about the same edit distance as learning identical structures, pooling the data has much worse performance both in terms of edit distance and in terms of KL-divergence.

For a qualitative perspective, Figure 2.6 shows the true structures and the structures learned by multitask learning and single-task learning for the five tasks (one per row) on one trial of the ALARM-COMP problem. The figure clearly shows that multitask learning finds more accurate structures by taking advantage of the similarity between the five tasks, while still preserving some of the true differences between them.

Figure 2.7 shows the performance of single and multitask learning as the train

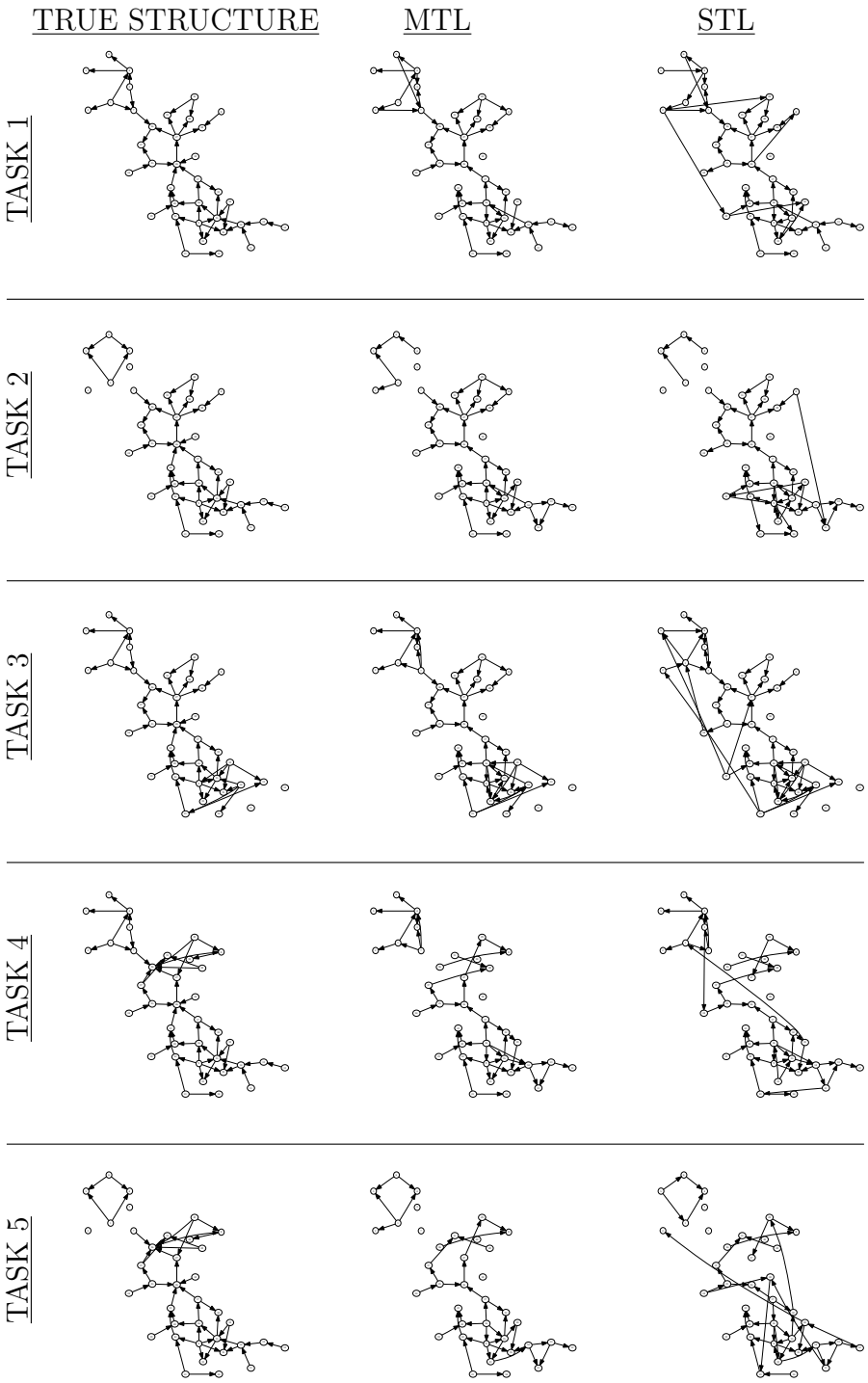


Figure 2.6: The true structures (left), structures learned by MTL (middle) and STL (right) for ALARM-COMP

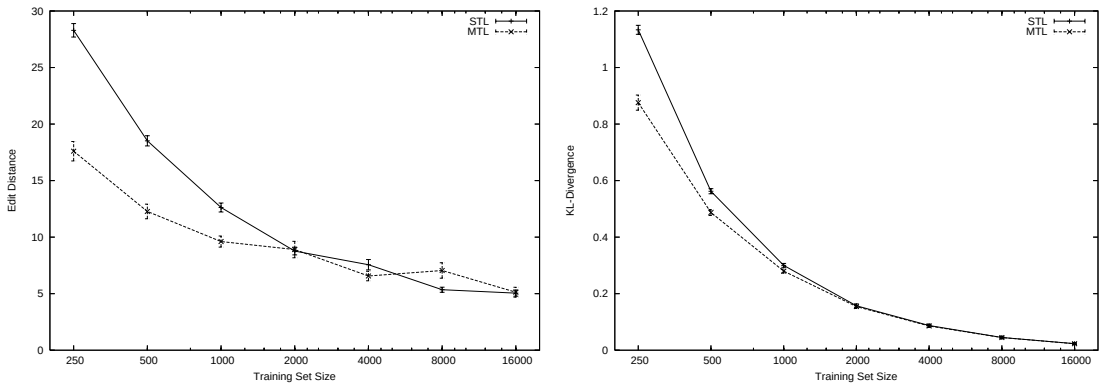


Figure 2.7: Edit distance (left) and KL-Divergence (right) vs. train set size for ALARM-COMP.

set size varies from 250 to 16000 cases (MTL uses 5% of the training points as a validation set to select the penalty parameter). As expected, the benefit from multitask learning is larger when the data is scarce and it diminishes as more training data is available. This is consistent with the behavior of multitask learning in other learning setting (see e.g. (Caruana, 1997)). For smaller training set sizes multitask learning needs about half as much data as single-task learning to achieve the same edit distance. In terms of KL-divergence, multitask learning provides smaller savings in sample size. One reason for this is that, as discussed before, multitask learning yields lower improvements in KL-divergence than in edit distance. For the most part however, the smaller savings in sample size are due to the fact that more training data leads not only to more accurate structures, but also to more accurate parameters. Since multitask structure learning only improves the structure and not the parameters, it is not able to *make up* for the loss of large amounts of training data.

The BIRD problem

The results on the BIRD problem mimic the ones in the previous section. Figure 2.8 shows the average (across the 6 BCRs/tasks) mean log likelihood on a large independent test set for multitask structure learning as a function of the penalty

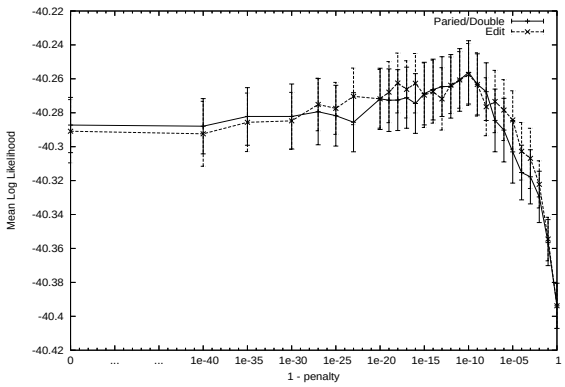


Figure 2.8: Average mean log likelihood vs. the penalty parameter for multitask structure learning on the BIRD problem.

parameter of the multitask prior. Each line corresponds to a different type of multitask prior. The x-axis plots $1 - penalty$, so the right most point corresponds to no penalty (single task learning) and the leftmost point corresponds to a penalty of one (learning identical structures). Higher mean log likelihood represents better performance. As with the five problems in the previous section, the type of multitask prior does not have a significant impact on the performance of multitask learning. As the penalty parameter increases ($1 - penalty$ decreases), information starts to be transfered between the different tasks and the performance quickly increases. After reaching a peak, the performance starts to decrease slowly as the penalty increases further.

Note that, because the tasks are not all defined on the same set of variables (see Section 2.3.4), the algorithm for learning identical structures for all tasks from Section 2.3.1 can not be directly applied. Our algorithm on the other hand can handle this situation and learns a set of identical structures for all tasks that performs reasonably well (left end of Figure 2.8).

Figure 2.9 shows the average mean log likelihood performance of multitask structure learning and single task structure learning as a function of the training set size. multitask learning uses 5% of the training data to select the penalty pa-

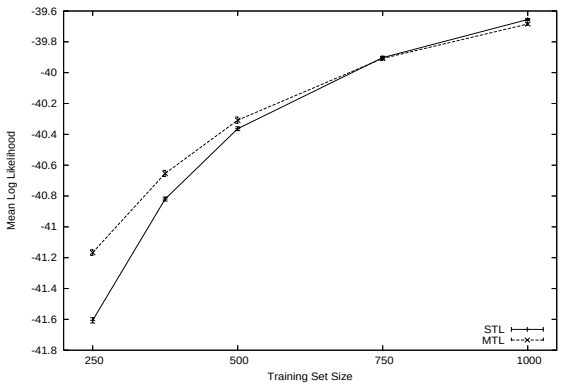


Figure 2.9: Average mean log likelihood vs. training set size for the BIRD problem.

rameter for the multitask prior. As with the other problems, the benefit from multitask learning is larger for smaller training set sizes. As the training size increases single-task learning catches up and eventually outperforms multitask learning. Unfortunately, since we do not know the real network structures for this problem, we can not directly asses the quality of the learned structures. The results in the previous sections, however, suggest that if one would be able to measure the edit distance between the true and the learned structures, the improvement provided by multitask learning in terms of edit distance probably would be even larger than the improvement provided in terms of average mean log likelihood.

2.4 Task Selection for Multitask Structure Learning

In any multitask learning setting there is a trade-off between *correct transfer* of features that are truly common between tasks, and *incorrect transfer* of features that in reality are distinct. More correct transfer translates into more benefit from multitask learning while incorrect transfer diminishes or even eliminates this benefit.

In multitask structure learning this trade-off is controlled via the multitask prior. If the tasks are closely related (i.e. their true structures are very similar)

then there is less opportunity for incorrect transfer and the multitask prior can safely encourage more sharing between tasks leading to higher benefits. Conversely, if the tasks are more dissimilar then incorrect transfer becomes more of a concern and the sharing between tasks needs to be toned down. The problem arises when some of the tasks are closely related, but others are quite dissimilar. In this case, multitask structures learning is either forced to lower the sharing between tasks and miss some opportunities for correct transfer from the closely related tasks, or to suffer from incorrect transfer from the dissimilar tasks. In both cases, the performance of multitask structure learning will be lower than if it were to only use the closely related tasks.

In what follows, we slightly change the problem setup, and assume that there is a single *principal task* that we are interested in learning a good network structure for. If more than one task is important, then the procedure can be repeated with each important task as the principal task. We also assume that there exists a pool of potentially related extra tasks, but we are not interested in their network structures. Under these assumptions, the goal of task selection is to find a set of tasks that, when used as extra tasks in multitask structure learning, maximize the performance of the principal task.

As discussed above, the more related a task is to the principal task, the higher the benefit it provides. This justifies the following task selection procedure: first order all the extra tasks by a measure of their relatedness to a principal task, then select the first (most related) N tasks to use as related tasks. N, the number of tasks to be selected, can either be specified by the user, or selected using an independent validation set.

This task selection procedure relies on having a measure of task relatedness. In the rest of the section we propose two such task relatedness measures.

2.4.1 A Direct Measure of Task Relatedness

Unlike most multitask learning settings, where the notion of task relatedness is not well defined, in the multitask structure learning setting there is a clear definition of relatedness: two tasks are related if they have similar structures. So any measure of similarity/dissimilarity between the true Bayes Net structures of two tasks provides a direct measure of the similarity/dissimilarity between the two tasks.

Since the true network structures are not available, we need to approximate the similarity between two tasks without having access to the true structures themselves. A simple way to do this is to first learn the network structures for each task from the training data (in a single-task manner), then use the learned networks to compute the similarity between the different tasks. One potential problem with this approach is that the greedy search procedure used to learn the network structures is a high variance procedure. This variance might make the learned networks artificially dissimilar leading to a poor, high variance, approximation of the similarity between different tasks.

This problem can be alleviated by encouraging the greedy search procedure to follow similar search paths for all task. Luckily, multitask structure learning does just that: it gives greedy search an incentive to make similar decisions for every task, making the search paths for all tasks similar. This incentive, quantified by the penalty parameter of the multitask prior, should be large enough to cut down the variance, but small enough not to make the learned structures artificially similar. In our experiments using a penalty parameter of 0.9 worked well.

To put it all together, the procedure we propose for measuring task relatedness consists of the following two steps:

1. Learn the structures for all the tasks using multitask structure learning with a small penalty parameter.

2. For each pair of tasks, use the similarity between their learned structures as a measure of similarity between the two tasks.

Note that the multitask structure learning at step 1 is only used as a pre-processing step. The structures learned are used only to compute the similarity between the tasks.

The measure of structure similarity/dissimilarity that is used to evaluate the task relatedness should reflect, if possible, the same prior beliefs about how the structures should be shared that are encoded in the multitask structure learning prior. For example if the prior belief is that part of the network structure should not be shared (e.g. the prior in Section 2.3 have a penalty of 0 for some of the arcs) then the similarity/dissimilarity measure should also ignore the respective part of the structure. In our experiments, we used the edit distance between two structures as a measure of dissimilarity between structures.

An added bonus of this method of assessing task relatedness is that it generates a proper distance metric between tasks. This is a desired property, especially if one wants to cluster the tasks rather than just order them.

2.4.2 An Indirect Measure of Task Relatedness

Another measure of task relatedness can be obtained using the following procedure:

1. Learn the structure of the first task using only data from the first task.
2. Keeping the structure fixed, learn the parameters using only data from the second task.
3. Compute the mean log likelihood of an independent validation set from the second task.

Steps 2 and 3 can of course be replaced with a cross-validation procedure in order to obtain a more accurate estimate of the mean log likelihood performance. In our experiments we use ten-fold cross validation, and use the average mean log likelihood over the ten folds as a measure of task relatedness.

This task relatedness measure is based on the intuition that the more related the tasks are, the more the network structure of one task should be compatible with the other task leading to a better mean log likelihood. Measuring task relatedness this way however, has a number of disadvantages when compared to the direct measure proposed in the previous section: it does not measure quite the right thing, it is not a metric (it is not even symmetric) and it can not be adapted to reflect different prior beliefs about how the structures should be shared.

2.4.3 Empirical Evaluation

We test the task selection method using a variation of the ALARM-COMP problem, and the bird ecology data.

Given the number of tasks to be selected, N, we evaluate the performance of multitask structure learning with task selection using the following procedure:

1. Compute the similarity between each extra task and the principal task.
2. Select the most similar N tasks to use as related tasks.
3. Learn the network structure of the principal task and the N selected tasks using multitask structure learning.
4. Report the Edit Distance and KL-Divergence between the learned network and the true network for the principal task.

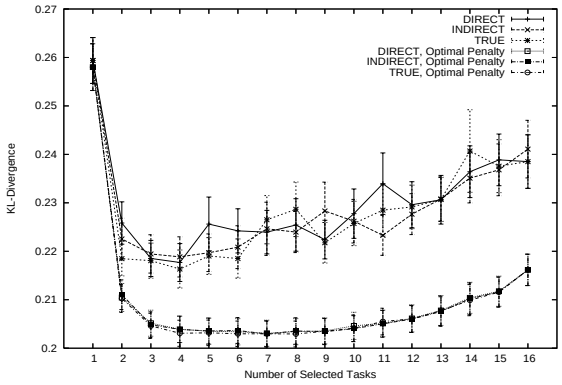


Figure 2.10: KL-Divergence of the principal task vs. the number of selected tasks for the ALARM-COMP problem.

The ALARM-COMP problem

As a first test, we attempt to recover the structure of the original ALARM network. The pool of extra tasks consist of all fifteen tasks that can be generated using the ALARM-COMP method described in Section 2.3.4: four tasks where only one of the four components is changed, six tasks where two components are changed, four tasks where three components are changed, and one task where every component is changed. Note that the arcs between components are never changed (they are the same as in the original ALARM network), so all the extra tasks have some degree of similarity to the principal task. All results in this section are averages across fifty random trials.

Figure 2.10 shows the KL-Divergence between the true and the learned networks for the principal task (the original ALARM network) as N, the number of tasks to be selected by the task selection procedure, varies from 0 (single task learning) to 15 (no task selection) for a training set of 1000 points. The two groups of lines in the graph show the performance when the penalty parameter for the multitask prior is selected using a small validation set of 50 points (the upper group), and when the penalty parameter is selected optimally (the lower group). Each group has three lines corresponding to three different measure of task re-

latedness: the direct measure from Section 2.4.1(DIRECT), the indirect measure from Section 2.4.2(INDIRECT), and the true task relatedness computed using the true network structures (TRUE).

The shape of all the lines is exactly as expected. At first, the performance of multitask learning increases as the most similar tasks are added to the set of related tasks. Then, when the tasks added to the set of related tasks become too dissimilar, incorrect transfer starts to be an issue and the performance decreases. This demonstrates that task selection is successful at improving the performance of multitask structure learning. Comparing the lines for the different task relatedness measures, we see that performance for both the direct and indirect measures is very similar to the performance obtained using the true task relatedness, indicating that both measures do a good job at ordering the tasks, and that, for this problem, there is nothing to gain by having a more accurate measure of task relatedness. Except for the lower variance and better performance of the results using optimal penalty selection, the two groups of lines have similar behavior suggesting that the above observations will hold regardless of how the prior parameter is selected. The gap between the two groups show that there is room to improve the multitask structure learning performance by having a better procedure for selecting the penalty parameter for the multitask prior.

Figure 2.11 shows the improvement in Edit Distance provided by multitask learning over single-task learning as a function of the size of the training set. Since single-task learning has no parameters that need to be set, it uses all the available data for training. The different lines in the graph correspond to different ways to select how many tasks should be included in the set of related tasks. As a general trend, the improvement provided by multitask learning over single-task learning diminishes as the training set becomes larger.

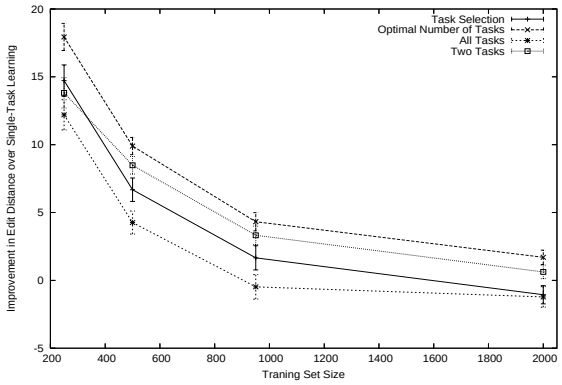


Figure 2.11: Improvement in Edit Distance over single task learning vs. training set size for the ALARM-COMP problem.

A simple method for selecting when to stop adding tasks to the related tasks set is to use a small validation set. This validation set should be different from the one used to select the multitask prior penalty parameter. The "Task Selection" line in Figure 2.11 shows the improvement in performance over single task learning for when 5% of the training data is used to select the number of tasks, and a separate 5% is used to select the penalty parameter. Compared to the "All Tasks" line that represents the performance of multitask structure learning when all tasks are used as related tasks (i.e. no task selection is done), task selection always yields an additional increase in performance, except for training set sizes of 2000 points when the performance is similar. To keep the comparison fair, multitask structure learning with no task selection uses 10% of the training data to select the penalty parameter for the multitask prior.

The "Optimal Number of Tasks" line depicts the performance of task selection when the size of the related tasks set is selected optimally. (The penalty parameter is still selected using 5% of the training data.) Being able to correctly decide when to stop adding tasks to the set of related tasks more than doubles the benefit of task selection (over multitask learning without task selection). Note that here we are using one of the simplest possible procedures for selecting the number of tasks.

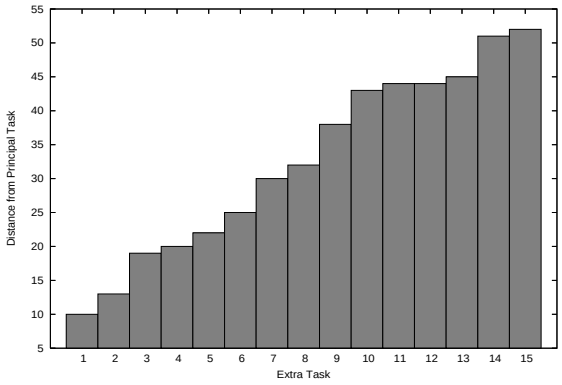


Figure 2.12: Distance between the principal task and the rest of the tasks for one trial of the ALARM-COMP problem. Distance is computed using the direct measure.

Improving the selection procedure is very likely to close the gap between the "Task Selection" line and the "Optimal Number of Tasks" line. We are currently running experiments using ten fold cross-validation to both select the penalty parameter for the multitask prior (which will move all the lines in Figure 2.11 up, widening the gap between them and single-task learning) and select the number of tasks (which will move only the "Task Selection" line up, widening the gap between it and the "All Tasks" line).

The task selection procedure used until now is entirely automatic, with no user input whatsoever. It is not unreasonable, however, to assume that the user is able to assist the task selection procedure. For example, Figure 2.12 shows the distance between each extra task and the principal task as computed using the direct method in Section 2.4.1 (tasks are sorted by the computed distance). A user could note that there is a gap between the distance of the second task and the distance of the third task and decide to only use the first two tasks as related tasks for multitask structure learning. Such "gap hunting" is commonly done in practice in areas such as unsupervised learning. The line labeled "Two Tasks" in Figure 2.11 shows the performance of multitask structure learning if the user decides to use only the closest two tasks as related tasks.

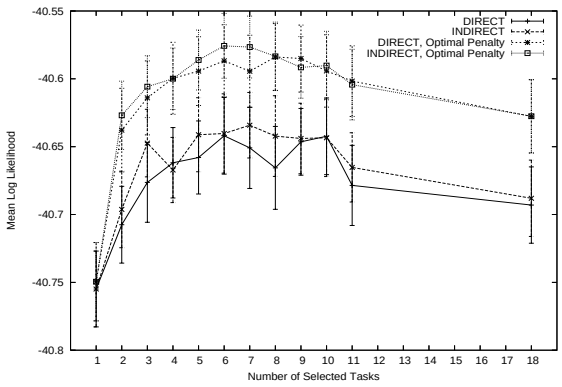


Figure 2.13: Mean log likelihood vs. the number of selected tasks for BCR28 on the BIRD problem.

The BIRD problem

In this section we use as the set of available tasks the eighteen BCRs that have more than 2500 instances after eliminating missing values: 30, 29, 28, 22, 13, 23, 5, 14, 24, 27, 32, 9, 10, 12, 16, 18, 21 and 31. To ensure accurate estimates of generalization performance, we only use as principal tasks (and therefore report results on) the first eleven of these BCRs, that have more than 8000 points. All results in this section are averages across twenty trials.

Figure 2.13 shows the mean log likelihood on a large independent test set when BCR 28 is used as the principal task. On the x-axis the number of related tasks selected by the task selection procedure varies from 0 (single-task learning) to 17 (no task selection). Higher mean log likelihood represents better performance. These results are obtained for a training set of 500 points. A validation set of 25 points is used to select the penalty parameter of the multitask prior for the lines labeled "DIRECT" and "INDIRECT". The other lines in the graph show the performance for the optimal penalty parameter for the direct and the indirect task relatedness measures. The graph depicts a similar story as for the ALARM-COMP problem; in the beginning the performance increases as the more similar tasks are added to the related tasks set, then the performance deteriorates as more dissimilar

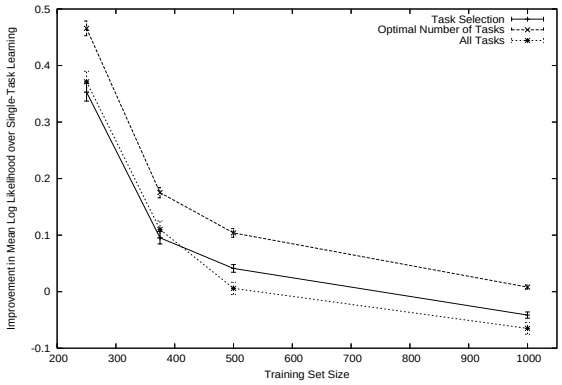


Figure 2.14: Average improvement in mean log likelihood over single task learning vs. training set size for task selection on the BIRD problem (average over 11 BCRs).

tasks are selected. Again, the performance for both task relatedness measures is similar.

Figure 2.14 shows the improvement in mean log likelihood of multitask structure learning over single-task structure learning as a function of training set size. The thee lines in the graph correspond to task selection with optimal number of tasks, task selection with the number of tasks selected using 5% of the training data, and no task selection. To select the penalty parameter for the prior, the two task selection methods use 5% of the training data and multitask learning without task selection uses 10% of the data. Single-task learning uses all available data for training. The task selection procedure is repeated using each of the eleven BCRs with most data as the principal task and the rest of the seventeen BCRs as extra tasks. The figure shows average improvement in mean log likelihood across the eleven BCRs.

For small training set sizes, multitask learning with task selection performs the same as multitask learning without task selection. When the training set gets larger, however, task selection improves the performance of multitask learning. As in the case of the ALARM-COMP problem, improving the procedure for deciding when to stop adding tasks to the set of related tasks would yield further improve-

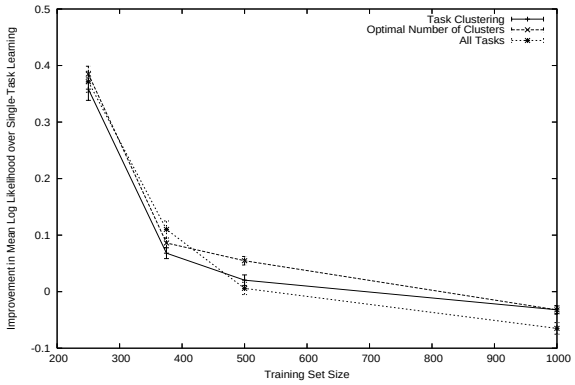


Figure 2.15: Average improvement in mean log likelihood over single task learning vs. training set size for task clustering on the BIRD problem (average over 11 BCRs).

ment. Improving the procedure for selecting a good value for the penalty parameter of the multitask prior also can lead to a significant increase in performance (see Figure 2.13).

An alternative to the task selection method we used until now, is to partition the tasks using a clustering algorithm and run the multitask structure learning algorithm on each partition. Here we clustered the tasks using hierarchical agglomerative clustering with average link[8]. Distance is measured using the direct task relatedness measure. To decide on the number of clusters we used a small validation set (the same validation set used by task selection to decide how many tasks to add to the set of related tasks).

Figure 2.15 shows the improvement in average (over the eleven BCRs) mean log likelihood provided by multitask structure learning with task clustering over single task learning as a function of training set size. The three lines in the figure correspond to task clustering where the number of clusters is selected using 5% of the training data as a validation set (Task Clustering), when the number of clusters is chosen optimally (Optimal Number of Clusters), and when using all

[8]At each step the clusters that have the smallest average distance between their members are joined

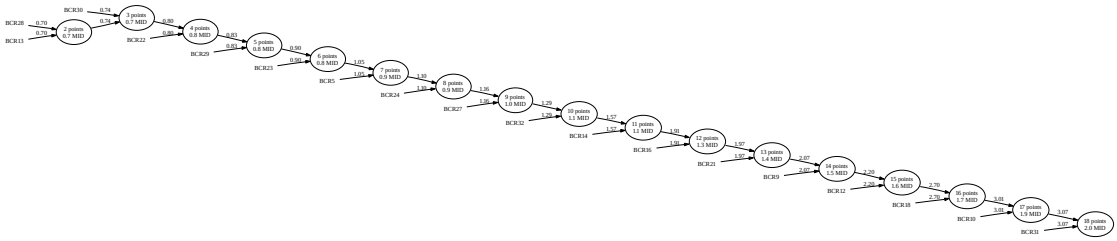


Figure 2.16: Cluster hierarchy for the BIRD problem.

tasks (i.e. having a single cluster) (All Tasks). To generate the Task Clustering and Optimal Number of Clusters lines, we use 5% of the training data to select the penalty parameter of the multitask prior (different from the 5% used to select the number of clusters). All Tasks uses 10% of the training data to select the penalty parameter.

Comparing the Task Clustering and the All Tasks lines, we see that performing task clustering yields worse performance than using all tasks for small training set sizes[9], but it does better for higher training set sizes. Comparing the Task Clusterings and the Optimal Number of Clusters lines we note that, in the case of task clustering, the difference in performance between selecting the number of clusters optimally, and selecting it using a validation set is smaller than in the case of task selection. Overall, however, the performance of task clustering is lower than the performance of task selection for this problem (see Figure 2.14). This is true both when selecting the number of tasks/number of clusters using a small validation set, and especially when selecting it optimally.

The reason why task clustering has lower performance is that, for this problem, the tasks do not form a well defined cluster structure. Figure 2.16 shows the clustering hierarchy (dendrogram) for one trial of the BIRD problem when a training set of 500 points was used to compute the distances between tasks. The dendrogram is laid on it's side, with the root on the right and the leafs toward

[9]The performance for All Tasks can be higher than the performance for Optimal Number of Clusters because All Tasks uses more data to select the penalty parameter.

the left. The cluster hierarchy is linear indicating that the tasks do not cluster together, but rather there is a single cluster and tasks keep getting added to it. Because task selection is more flexible than task clustering, it is able to handle this situation better and obtain higher performance.

Examining the dendrogram more closely, we see that the first six BCRs (i.e. 30, 29, 28, 22, 13, 23) form a somewhat tighter group, with the rest of the task being increasingly further away (both from the group of six and from each other). This prompted us to use the first six BCRs as related tasks when evaluating the performance of multitask structure learning in Section 2.3.4.

2.5 Discussion and Related Work

multitask learning has been applied to a wide variety of learning methods. The work most closely related to ours is Baxter's (Baxter, 1997) which provides a Bayesian interpretation of multitask learning. multitask learning has also been used in the context of neural networks (e.g. (Caruana, 1997)), kernel learning (e.g. (Jebara, 2004)), Gaussian processes (e.g. (Lawrence & Platt, 2004; Yu et al., 2005)), and Dirichlet processes (e.g. (Teh et al., 2006)) to name a few.

Learning the structure of Bayes Nets from data has also received a lot of attention in he past years. For an overview of traditional learning methods see (Cooper & Hersovits, 1992; Heckerman, 1999; Buntine, 1996; Spirtes et al., 2000). In this chapter, we use heuristic search in the space of network structures. Some straightforward extensions are greedy search in the space of equivalence classes (Chickering, 1996), obtaining confidence measures on the structural features of the configurations via bootstrap analysis (Friedman et al., 1999), and structure learning from incomplete datasets via the structural EM algorithm (Friedman, 1998). Other extensions such as obtaining a sample from the posterior distribu-

tion via MCMC methods might be more problematic. Because of the larger search space, MCMC methods might not converge in reasonable time. Evaluating different MCMC schemes is a direction for future work. Another open question is whether we can relax the requirement that the parameters of the Bayes Nets for the different related tasks are independent *a priori*. Relaxing this requirement might further improve the performance of multitask learning since the task would be able to share not only the structures but also the parameters, thus having more opportunities for inductive transfer.

One interesting domain where multitask structure learning might prove useful is learning Bayesian multi-nets (Friedman et al., 1997). In Bayesian multi-nets a special attribute is selected (usually the class attribute), and a separate network is learned for each value of that attribute. To the best of our knowledge, all work in learning Bayesian multi-nets treats each separate network as an independent learning problem, in a single-task manner. Since it is reasonable to assume that the networks for the different values of the class attribute should be similar, learning all the networks simultaneously using multitask structure learning might yield improved performance.

There are relatively few papers that tackle the task selection/task weighting problem. Thrun and O'Sullivan (1996) proposed the task clustering (TC) algorithm in the context of multitask Nearest Neighbor algorithm. The task similarity function they used to perform the clustering is similar in spirit to the indirect measure of task relatedness in Section 2.4.2. Bakker and Heskes (2003) also developed a task clustering method for multitask Bayesian neural networks. Neither of these papers consider the problem of selecting the number of clusters. They both assume that the number of clusters is specified by the user. Silver and Mercer (1998) propose a method for dynamically adjusting the weights of the extra tasks

as the training of a multitask neural network progresses. Their method also has one free parameter that must be specified by the user.

A few authors provided a theoretical treatment for different task relatedness measures in restricted cases. Ben-David and Schuller (2003) provide a formal framework for task relatedness in a restricted setting where the data for all tasks is generated from the same underlying distribution but it is transformed using a different, unknown, function for each task. Juba (2006) presents a characterization task relatedness in terms of Kolmogorov complexity.

2.6 Conclusions

We present a method for learning the Bayes Net structures of related tasks. The approach assumes that the structures of related tasks are similar: the presence or absence of arcs in some of the structures provides evidence for the presence or absence of those same arcs in the other structures. When this assumption is true, learning the structures together yields an advantage over learning the structure for each task individually. Experiments with perturbed ALARM and INSURANCE networks and a real bird ecology problem show that learning related structures simultaneously via multitask structure learning yields significantly better performance when compared to learning the structures independently, especially when training data is scarce.

We also present a solution to the task selection problem for multitask structure learning. Our solution takes advantage of the fact that, unlike in most other multitask learning settings, in multitask structure learning there is a clear definition of task relatedness: two tasks are related if they have similar structures. We propose two methods for measuring task relatedness from data and use them as the basis of an effective task selection procedure. When dealing with tasks with different de-

grees of relatedness our task selection procedure further improves performance by selecting an appropriate set of tasks to be used by the multitask structure learning algorithm.

Acknowledgments

This is joint work with Rich Caruana. Thanks to Art Munson and the collaborators at Cornell Lab of Ornithology for the help with the bird ecology data. Thanks to Joe Halpern and John Platt for helpful comments. This work was supported by NSF Awards 0412930, 0347318 0427914, and 0612031.

BIBLIOGRAPHY

Bakker, B., & Heskes, T. (2003). Task clustering and gating for bayesian multitask learning. *The Journal of Machine Learning Research*, *4*, 83–99.

Baxter, J. (1997). A bayesian/information theoretic model of learning to learn via multiple task sampling. *Mach. Learn.*, *28*, 7–39.

Beinlich, I., Suermondt, H., Chavez, R., & Cooper, G. (1989). The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *Proceedings of the Second European Conference on Artificial Intelligence in Medicine*.

Ben-David, S., & Schuller, R. (2003). Exploiting task relatedness for multiple task learning. *Proc. of the 16th Annual Conference on Learning Theory*.

Binder, J., Koller, D., Russell, S., & Kanazawa, K. (1997). Adaptive probabilistic networks with hidden variables. *Machine Learning*, *29*.

Buntine, W. (1991). Theory refinement on bayesian networks. In *Proc. 7th conference on uncertainty in artificial intelligence (uai '91)*.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Trans. On Knowledge and data Engineering*, *8*, 195–210.

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*, 41–75.

Chickering, D. (1996). Learning equivalence classes of Bayesian network structures. *Proc. 12th Conference on Uncertainty in Artificial Intelligence (UAI'96)*.

Cooper, G., & Hersovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Maching Learning*, *9*, 309–347.

Friedman, N. (1998). The Bayesian structural EM algorithm. *Proc. 14th Conference on Uncertainty in Artificial Intelligence (UAI '98)*.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian Network Classifiers. *Machine Learning*, *29*, 131–163.

Friedman, N., Goldszmidt, M., & Wyner, A. J. (1999). Data analysis with bayesian networks: A bootstrap approach. *Proc. 15th Conference on Uncertainty in Artificial Intelligence*.

Friedman, N., Linial, M., Nachman, I., & Pe'er, D. (2000). Using bayesian networks to analyze expression data. *J. Comput. Biol.*, *7*, 601–620.

Heckerman, D. (1999). A tutorial on learning with bayesian networks. *Learning in graphical models*, 301–354.

Heckerman, D., Mamdani, A., & Wellman, M. (1995). Real-world applications of Bayesian networks. *Communications of the ACM*, *38*, 24–30.

Jebara, T. (2004). Multi-task feature and kernel selection for svms. *ICML '04: Twenty-first international conference on Machine learning*.

Juba, B. (2006). Estimating relatedness via data compression. *Proc. of the 23rd International Conference on Machine learning* (pp. 441–448). ACM Press New York, NY, USA.

Lawrence, N. D., & Platt, J. C. (2004). Learning to learn with the informative vector machine. *ICML '04: Twenty-first international conference on Machine learning*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Silver, D., & Mercer, R. (1998). The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Learning to learn*, 213–233.

Spirtes, P., Glymour, C., & Scheines, R. (2000). *Causation, prediction, and search*. Cambridge, MA: The MIT Press. Second edition.

Teh, Y., Jordan, M., Beal, M., & Blei, D. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems*.

Thrun, S., & O'Sullivan, J. (1996). Discovering structure in multiple learning tasks: The TC algorithm. *In Proceedings of the 13th International Conference on Machine Learning*. Morgen Kaufmann.

Yu, K., Tresp, V., & Schwaighofer, A. (2005). Learning Gaussian Processes from Multiple Tasks. .

CHAPTER 3

PREDICTING GOOD PROBABILITIES WITH SUPERVISED LEARNING

3.1 Introduction

In many applications it is important to predict well calibrated probabilities; good accuracy or area under the ROC curve are not sufficient. This chapter examines the probabilities predicted by ten supervised learning algorithms: SVMs, neural nets, decision trees, memory-based learning, bagged decision trees, random forests, boosted decision trees, boosted decision stumps, Naive Bayes and logistic regression. We show how maximum margin methods such as SVMs, boosted trees, and boosted stumps tend to push predicted probabilities away from 0 and 1. This hurts the quality of the probabilities they predict and yields a characteristic sigmoid-shaped distortion in the predicted probabilities. Other methods such as Naive Bayes have the opposite bias and tend to push predictions closer to 0 and 1. And yet other learning methods such as bagged trees, logistic regression and neural nets have little or no bias and predict well-calibrated probabilities on most problems.

After examining the distortion (or lack thereof) characteristic to each learning method, we experiment with two calibration methods for correcting these distortions.

Platt Scaling: a method for transforming the predictions of learning methods to posterior probabilities by passing them through a logistic function. This method has been developed by Platt (1999) to calibrate SVM outputs.

Isotonic Regression: the method used by Zadrozny and Elkan (2002; 2001) to calibrate predictions from boosted Naive Bayes, SVM, and decision tree models.

Platt Scaling is most effective when the distortion in the predicted probabilities is sigmoid-shaped. Isotonic Regression is a more powerful calibration method that can correct any monotonic distortion. Unfortunately, this extra power comes at a price. A learning curve analysis shows that Isotonic Regression is more prone to overfitting, and thus performs worse than Platt Scaling, when data is scarce. For boosted trees we also examine Logistic Correction, a calibration method suggested by boosting theory, and boosting to log-loss instead of exponential loss.

Finally, we examine how good are the probabilities predicted by each learning method after each method's predictions have been calibrated. Experiments with eleven classification problems suggest that bagged decision trees, random forests and neural nets trees predict the best probabilities prior to calibration, but after calibration the best methods are boosted trees, random forests and SVMs.

3.2 Calibration Methods

In this section we describe two methods for mapping model predictions to posterior probabilities: Platt Calibration and Isotonic Regression. Both of these methods are designed for binary classification. See Zadrozny and Elkan (2002) for one way to deal with multiclass problems. In this chapter we consider only the binary case.

3.2.1 Platt Calibration

Platt (1999) proposed transforming SVM predictions to posterior probabilities by passing them through a sigmoid. We will see in Section 3.3 that a sigmoid transformation is also justified for boosted trees and boosted stumps.

Let the output of a learning method be $f(x)$. To get calibrated probabilities, pass the output through a sigmoid:

$$P(y = 1|f) = \frac{1}{1 + exp(Af + B)} \tag{3.1}$$

where the parameters A and B are fitted using maximum likelihood estimation from a fitting training set (f_i, y_i). Gradient descent is used to find A and B such that they are the solution to:

$$\underset{A,B}{argmin}\{-\sum_i y_i log(p_i) + (1 - y_i) log(1 - p_i)\}, \tag{3.2}$$

where

$$p_i = \frac{1}{1 + exp(Af_i + B)} \tag{3.3}$$

Two questions arise: where does the sigmoid train set come from? and how to avoid overfitting to this training set?

If we use the same data set that was used to train the model we want to calibrate, we introduce unwanted bias. For example, if the model learns to discriminate the train set perfectly and orders all the negative examples before the positive examples, then the sigmoid transformation will output just a 0,1 function. So we need to use an independent calibration set in order to get good posterior probabilities. This, however, is not a draw back, since the same set can be used for model and parameter selection.

To avoid overfitting to the sigmoid train set, an out-of-sample model is used. If there are N_+ positive examples and N_- negative examples in the train set, for each training example Platt Calibration uses target values y_+ and y_- (instead of 1 and 0, respectively), where

$$y_+ = \frac{N_+ + 1}{N_+ + 2};\ y_- = \frac{1}{N_- + 2} \tag{3.4}$$

For a more detailed treatment, and a justification of these particular target values see (Platt, 1999).

3.2.2 Isotonic Regression

The sigmoid transformation works well for some learning methods, but it is not appropriate for others. Zadrozny and Elkan (2002; 2001) successfully used a more general method based on Isotonic Regression (Robertson et al., 1988) to calibrate predictions from SVMs, Naive Bayes, boosted Naive Bayes, and decision trees. This method is more general in that the only restriction is that the mapping function be isotonic (monotonically increasing). That is, given the predictions f_i from a model and the true targets y_i, the basic assumption in Isotonic Regression is that:

$$y_i = m(f_i) + \epsilon_i \tag{3.5}$$

where m is an isotonic (monotonically increasing) function. Then, given a train set (f_i, y_i), the Isotonic Regression problem is finding the isotonic function $\hat{m}$ such that

$$\hat{m} = argmin_z \sum (y_i - z(f_i))^2 \tag{3.6}$$

One algorithm that finds a stepwise constant solution for the Isotonic Regression problem is pool-adjacent violators (PAV) algorithm (Ayer et al., 1955) presented in Table 3.1.

As in the case of Platt calibration, if we use the model training set (x_i, y_i) to get the training set $(f(x_i), y_i)$ for Isotonic Regression, we introduce unwanted bias. So we use an independent validation set to train the isotonic function.

Table 3.1: PAV Algorithm

Algorithm 1. PAV algorithm for estimating posterior probabilities from uncalibrated model predictions.

1 Input: training set (f_i, y_i) sorted according to f_i
2 Initialize $\hat{m}_{i,i} = y_i$, $w_{i,i} = 1$
3 While $\exists\ i\ s.t.\ \hat{m}_{k,i-1} \geq \hat{m}_{i,l}$
  Set $w_{k,l} = w_{k,i-1} + w_{i,l}$
  Set $\hat{m}_{k,l} = (w_{k,i-1}\hat{m}_{k,i-1} + w_{i,l}\hat{m}_{i,l})/w_{k,l}$
  Replace $\hat{m}_{k,i-1}$ and $\hat{m}_{i,l}$ with $\hat{m}_{k,l}$
4 Output the piecewise constant function:
  $\hat{m}(f) = \hat{m}_{i,j}$, for $f_i < f \leq f_j$

3.3 Qualitative Analysis of Predictions

In this section we empirically examine the relationship between the predictions of ten different classes of learning algorithms and the class posterior probabilities. Instead of quantitatively measuring the calibration of the learning methods, in this section we will qualitatively analyze the predictions and identify consistent patterns in the relationship between the predicted values and the true conditional probabilities for different learning algorithms.

Ideally one would asses the calibration of a model by comparing it's predicted values to the true class conditional probabilities. Unfortunately, on real problems, only the class label is known, not the true conditional probability of that class given the input. In this situation model calibration can be visualized using reliability diagrams (DeGroot & Fienberg, 1982). First, the prediction space is discretized into bins. We will use ten bins here. Cases with predicted value between 0 and 0.1 fall in the first bin, between 0.1 and 0.2 in the second bin, etc. For each bin, the mean predicted value is plotted against the true fraction of positive cases. If the model is well calibrated the points will fall near the diagonal line. For examples of reliability diagrams see the bottom row of Figure 3.1.

Besides reliability diagrams, we also show histograms of the values predicted by models trained with different learning algorithms, both before and after post training calibration. The histogram plots give insight into the behavior of the algorithms and help explain specific distortions we see in the reliability diagrams. For examples of prediction histogram plots see the top row in Figure 3.1.

To generate the figures in this section, for each class of algorithm we trained a large number of classifiers using different variations of the algorithm and parameter settings. For example, we train models using ten decision tree styles, neural nets of many sizes, SVMs with many kernels, etc. All the variations and the parameter settings we used for the different learning methods are described in Appendix 4.B. After training, we apply Platt Scaling and Isotonic Regression to calibrate all models. Each model is trained on the same random sample of 4000 cases and calibrated on independent samples of 1000 cases. The histograms of predicted values and the reliability diagrams are generated using a large independent test set that was not used for training or calibration. For the analysis we use eleven binary prediction problems with varying characteristics. A description of the problems can be found in Appendix 4.A.

Unless otherwise specified, the figures will show, for each learning algorithm, the histograms of predicted values and the reliability diagrams for the model trained with the respective algorithm that has the best squared error after post training calibration. So for each model class all graphs corresponding to the same problem are generated using the same model. For example, in Figures 3.2(a), 3.2(b), 3.2(c), 3.4(a), and 3.5(a) the graphs are generated using boosted MML trees after 1024 steps of boosting for the COV_TYPE problem, boosted SMML trees after 2048 steps for the LETTER.P1 problem and so on.

3.3.1 Boosting

In a recent evaluation of learning algorithms (Caruana & Niculescu-Mizil, 2006), boosted decision trees had excellent performance on metrics such as accuracy, lift, area under the ROC curve, average precision, and precision/recall break even point. However, boosted decision trees had poor squared error and cross-entropy because AdaBoost produces distorted probability estimates.

Friedman et al. (2000) provide an explanation for why boosting makes poorly calibrated predictions. They show that boosting can be viewed as an additive logistic regression model. A consequence of this is that the predictions made by boosting are trying to fit a logit of the true probabilities, as opposed to the true probabilities themselves. To get back the probabilities, the logit transformation must be inverted.

In their treatment of boosting as a large margin classifier, Schapire et al. (1998) observed that in order to obtain large margin on cases close to the decision surface, AdaBoost will sacrifice the margin of the easier cases. This results in a shifting of the predicted values away from 0 and 1, hurting calibration. This shifting is also consistent with Breiman's interpretation of boosting as an *equalizer* (see Breiman's discussion in (Friedman et al., 2000)). In what follows we demonstrate this probability shifting on real data.

Boosted Decision Trees

The particular boosting algorithm we evaluate in this chapter is AdaBoost.M1 (Schapire, 2001) with resampling. We boost both full decision trees and stumps as base level models. We believe the general results and observations in this chapter remain true when using other boosting algorithms and other base level models. To prevent the results from depending on one specific tree style, we boost

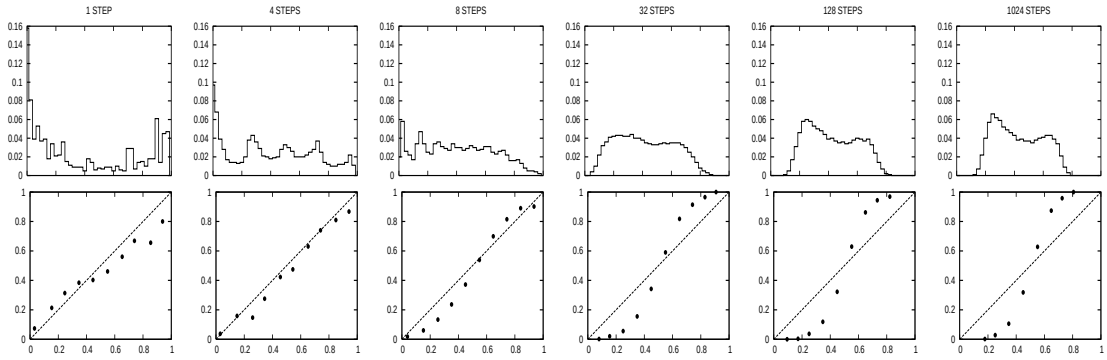
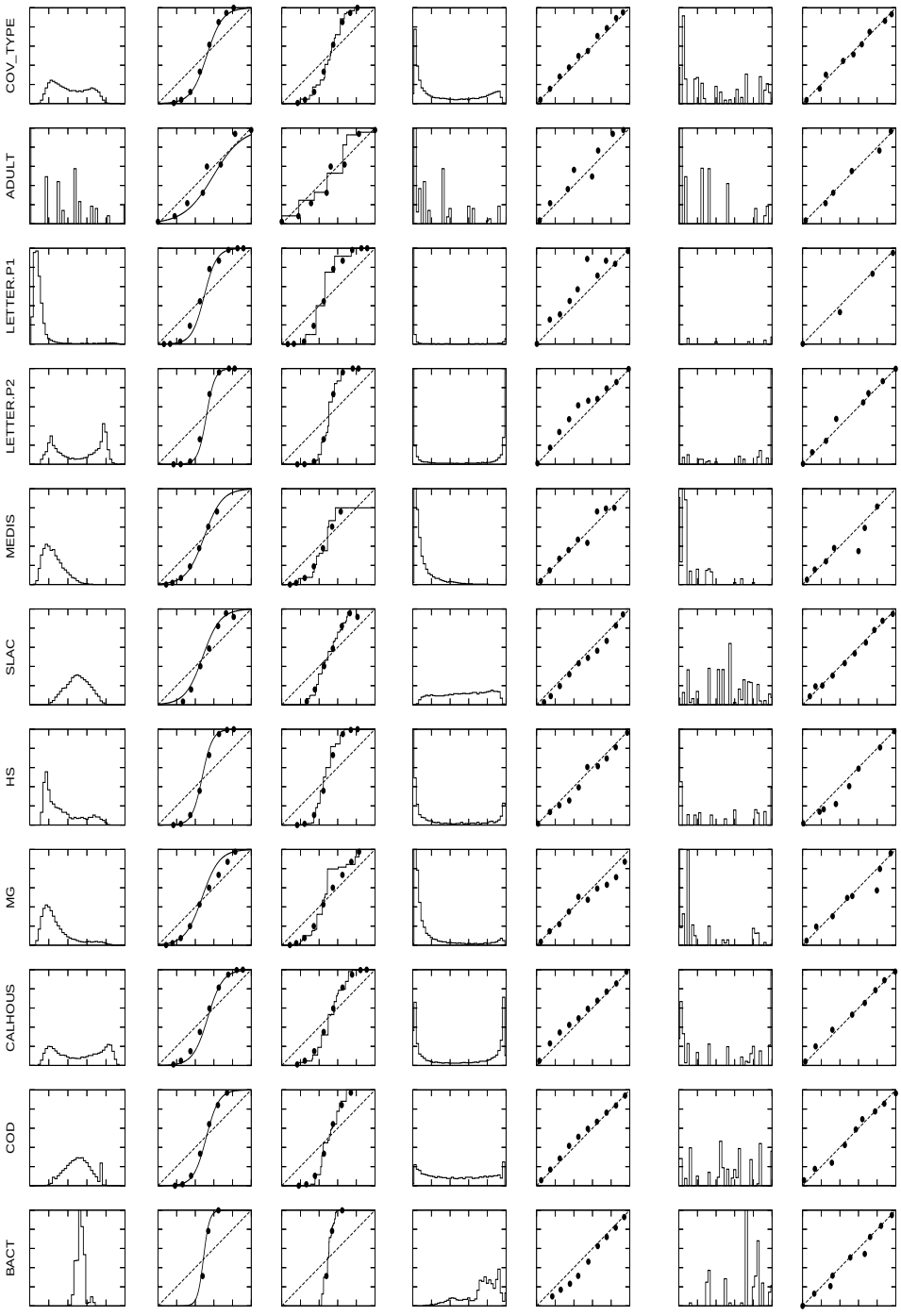


Figure 3.1: Effect of boosting on the predicted values. Histograms of the predicted values (top) and reliability diagrams (bottom) on the test set for boosted trees at different steps of boosting on the COV_TYPE problem.

ten different styles of trees. We use all the tree types in Buntine's IND package (1991) (ID3, CART, CART0, C4.5, MML, SMML, BAYES) as well as three new tree types that should predict better probabilities: unpruned C4.5 with Laplacian smoothing (Provost & Domingos, 2003); unpruned C4.5 with Bayesian smoothing; and MML trees with Laplacian smoothing. Because boosting can overfit (Rosset et al., 2004; Friedman et al., 2000), and because many iterations of boosting can make calibration worse (see Figure 3.1), we consider boosted tree models after 2,4,8,16,32,64,128,256,512,1024 and 2048 steps of boosting and select whichever iteration yields the best squared error.

The top row of Figure 3.1 shows histograms of the predicted values on a large test set after 1,4,8,32,128, and 1024 stages of boosting Bayesian smoothed decision trees (Buntine, 1992). The bottom row of the figure shows reliability diagrams for the same models. The histograms show that as the number of steps of boosting increases, the predicted values are pushed away from 0 and 1 and tend to collect on either side of the decision surface. This shift away from 0 and 1 hurts calibration and yields sigmoid-shaped reliability plots.

Figure 3.2(a) shows histograms of the predicted values (left column) and reliability diagrams (middle and right columns) for boosted trees on the eleven test



(a) Uncalibrated (b) Platt Scaling (c) Isotonic Regression

Figure 3.2: Histograms of predicted values and reliability diagrams for boosted decision trees before and after calibration.

problems on large test sets not used for training or calibration. The figure shows the same shift in predicted values away from 0 and 1 that generates the sigmoidal shaped reliability diagrams for ten out of the eleven problems. Because boosting overbite's on the ADULT problem, the best performance is achieved after only four iterations of boosting. If boosting is allowed to continue for more iterations, it will display the same sigmoidal shape on ADULT as in the other figures. The histograms of predicted values (top row in Figure 3.2(a)), show that almost all the values predicted by boosted trees lie in the central region with few predictions approaching 0 or 1. The one exception is LETTER.P1, a highly skewed data set that has only 3% positive class. On this problem some predicted values do approach 0, though careful examination of the histogram shows that even on this problem there is a sharp drop in the number of cases predicted to have probability near 0. This shift of predicted values away from 0 ad 1 is always accompanied by a sigmoid-shaped reliability diagram. It is this distinctive shape of the plots that motivates the use of a sigmoid to transform predictions into calibrated probabilities. The reliability plots in the middle row of the figure show sigmoids fitted using Platt's method. The reliability plots in the bottom of the figure show the function fitted with Isotonic Regression. Both calibration methods are able to closely fit the reliability diagrams indicating that after calibration with either method boosted decision trees will yield much better quality probabilistic predictions.

To show how calibration transforms predictions, we plot histograms and reliability diagrams for the eleven problems for boosted trees after Platt Calibration (Figure 3.2(b)) and Isotonic Regression (Figure 3.2(c)). The figures show that calibration undoes the shift in probability mass caused by boosting: after calibration many more cases have predicted probabilities near 0 and 1. The reliability diagrams are closer to diagonal, and the S-shape characteristic of boosted tree

predictions is now gone. On each problem, transforming predictions using Platt Scaling or Isotonic Regression yields a significant improvement in the predicted probabilities, leading to much lower squared error and log-loss. One difference between Isotonic Regression and Platt Scaling is apparent in the histograms: because Isotonic Regression generates a piecewise constant function, the histograms are coarse, while the histograms generated by Platt Scaling are smoother.

Boosted Decision Stumps

To asses the impact of the complexity of the base level model on calibration we also boost 1-level decision stumps. We boost five different types of stumps by using all of the splitting criteria in Buntine's IND package. Since boosting might converge slower when using weaker base-level models, we also consider boosted stumps models after 4096 and 8192 steps in addition to all the number of steps we consider for boosted decision trees.

The same observations hold when considering boosting weaker decision stumps instead of boosting full decision trees. Figure 3.3 shows the histograms of predicted values and the reliability diagrams for boosted decision stumps before calibration (3.3(a)), after Platt Scaling (3.3(b)) and after Isotonic Regression (3.3(c)) on three problems. Graphs for all eleven problems are included in Figure 3.14 in Appendix 3.A. Except for the BACT problem, uncalibrated boosted stumps models display an even more extreme shift in the predicted values toward the center of the plot than boosted decision trees. Interestingly, for the COV_TYPE problem the models used to generate the graphs (i.e. the models with the lowest squared error after calibration) were both obtained after 1024 steps of boosting for both boosted trees and boosted stumps. And for the MEDIS, SLAC, MG2, COD, and BACT problems the boosted stumps models used to generate the graphs were boosted for fewer steps than the corresponding boosted trees models (between 128

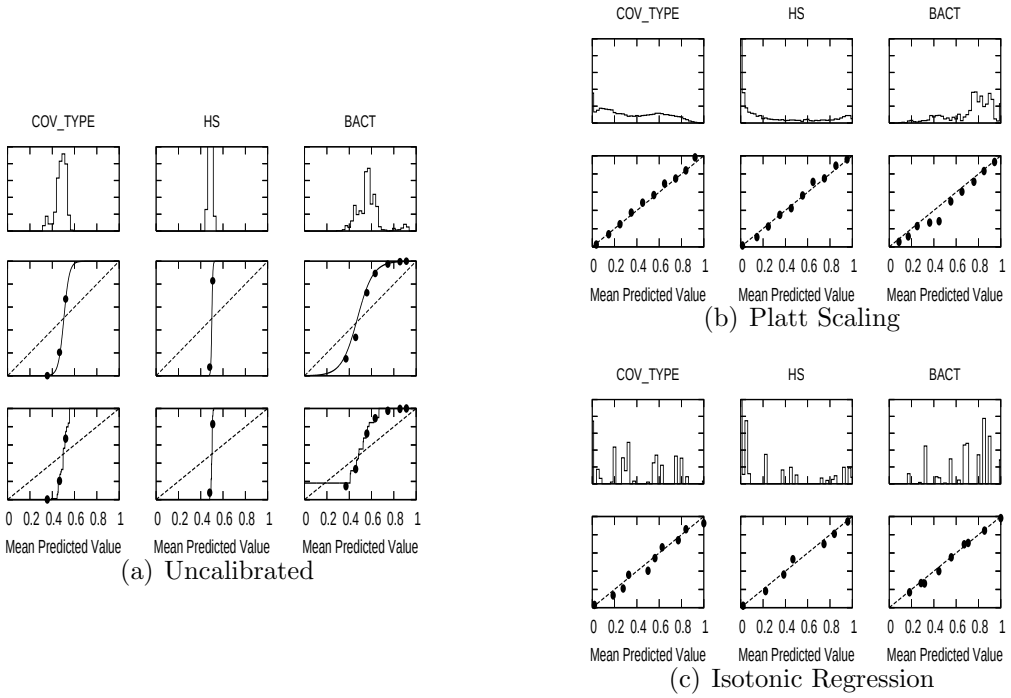
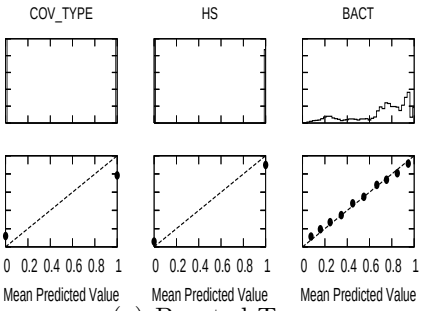


Figure 3.3: Histograms of predicted values and reliability diagrams for boosted decision stumps before and after calibration.

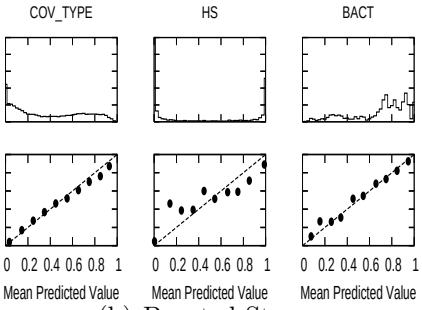
and 1024 steps for boosted stumps versus 2048 steps for boosted trees). This implies that the more extreme shift in the predicted values toward the center of the plot is mainly due to the lower expressive power of the decision stumps.

Logistic Correction

As mentioned before, in the process of optimizing the exponential loss function AdaBoost is attempting to fit the logit of the class conditional probabilities instead of the true probabilities themselves. This suggests that the true probabilities might be recovered by inverting the logit function fitted by AdaBoost. In this chapter we will refer to this alternative method for calibrating boosting's predictions as Logistic Correction. Like Platt Scaling, Logistic Correction transforms the predictions by passing them through a sigmoid, but the sigmoid parameters are analytically derived from Friedman et al's analysis of boosting as an additive logistic regres-

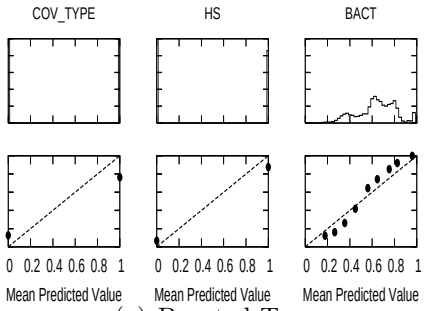


(a) Boosted Trees

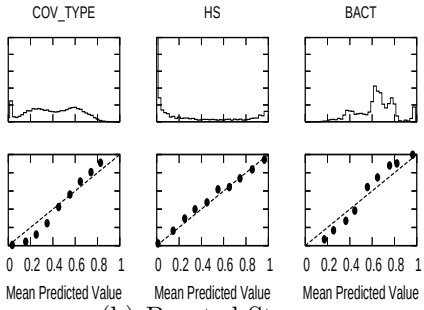


(b) Boosted Stumps

Figure 3.4: Histograms of predicted values and reliability diagrams for (a)boosted trees and (b)boosted stumps calibrated with Logistic Correction.



(a) Boosted Trees



(b) Boosted Stumps

Figure 3.5: Histograms of predicted values and reliability diagrams for (a)boosted trees and (b)boosted stumps trained to directly optimize log-loss.

sion model (Friedman et al., 2000) rather than being fitted using an additional calibration set.

Figure 3.4(a) shows prediction histograms and reliability diagrams for boosted decision trees after applying Logistic Correction for the COV_TYPE, HS and BACT problems. The prediction histograms show that for the COV_TYPE and HS data sets, AdaBoost with Logistic Correction predicts only zero or one, and thus has very poor calibration. For these data sets full decision trees are expressive enough to enable AdaBoost to perfectly separate the training set. After separating the training set, AdaBoost will keep pushing the predictions toward the tails of the sigmoid used by Logistic Correction generating the extreme predictions (Rosset et al., 2004). In our experiments this is a common result when applying Logistic

Correction to boosted decision trees, with ten out of the eleven data sets displaying this same behavior (see Figure 3.16(a) in Appendix 3.A).[1] The only exception is the BACT problem where the histogram shows a spread of predicted values, and the reliability diagram is close to the diagonal line indicating good calibration. It seems that on this problem even full decision trees are not powerful enough to allow AdaBoost to perfectly separate the training set.

When using weaker decision stumps as base level models, AdaBoost is no longer able to perfectly separate these training set, and the histograms show a wider range of predicted values than in the case of boosted full decision trees (see Figure 3.4(b) and Figure 3.15(a) in Appendix 3.A). With boosted stumps Logistic Correction yields well calibrated models that have reliability plots close to the diagonal.

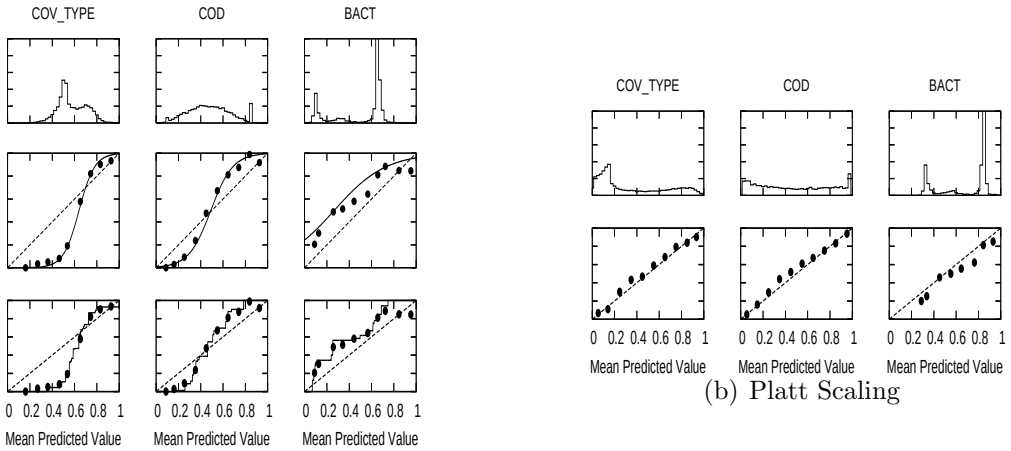
Logistic Correction has the advantage that it does not require an extra independent calibration set to fit the calibration models as is required for Platt Scaling and Isotonic Regression. The down side, however, is that Logistic Correction only seems to work reliably when boosting weaker models such as boosted decision stumps, models that have limited expressive power and are unable to capture the complexities of many real datasets. (See Section 3.4.) Experiments with boosting somewhat more complex two-level stumps indicate that on many data sets Logistic Correction already begins to overfit. There does not appear to be a decision tree model of intermediate complexity that, when boosted, works well with Logistic Correction for these problems.

[1]The histogram for the ADULT problem does not show these extreme zero-one predictions only because, as mentioned above, boosted trees overfit on this data set and the best model (which is plotted in the figure) is obtained after only four steps of boosting. If allowed to continue for more iterations the plots for ADULT are similar to the plots for the other nine problems.

Directly Optimizing Log-Loss

Another alternative to obtain well calibrated predictions from boosting is to use a variant of boosting that directly optimizes cross-entropy (log-loss) instead of the usual exponential loss. Collins et al. (2002) show that a boosting algorithm that optimizes log-loss can be obtained by simple modification to the AdaBoost algorithm. Collins et al. briefly evaluate this new algorithm on a synthetic data set, but acknowledge that a more thorough evaluation on real data sets is necessary. Lebanon and Lafferty (2001) shows that Logistic Correction applied to boosting with exponential loss should behave similarly to boosting with log-loss, and then demonstrate this by examining the performance of boosted stumps on a variety of data sets.

Our results confirm their findings for boosted stumps, and more importantly, also show the same effect for boosted trees. When optimizing log-loss, boosting decision trees generates models that make zero-one predictions on all problems except for BACT, while boosting decision stumps generates better calibrated models with a wide spread of predicted values (see Figure 3.5 and Figures 3.16(b) and 3.15(b) in Appendix 3.A). Compared to boosting with Logistic Correction, the reliability diagrams for boosting with log-loss are slightly further from the diagonal on the BACT problem for boosted trees and on all problems but HS for boosted stumps. This indicates that, when it works, optimizing to log-loss produces models that are slightly less well calibrated than models generated by applying Logistic Correction to boosting with exponential loss. Interestingly, for most problems, the reliability diagrams for boosting with log-loss still have a sigmoidal shaped, although it is not nearly as accentuated as in the case of uncalibrated boosted trees or stumps. Also, comparing the prediction histograms for Logistic Correction and optimizing to log-loss one can notice a small shift of predicted values away from zero and one.



(a) Uncalibrated

(b) Platt Scaling

Figure 3.6: Histograms of predicted values and reliability diagrams for SVMs before and after calibration.

Like Logistic Correction, optimizing to log-loss has the advantage that no extra data is required for calibration at the expense of requiring less expressive models in order to work reliably.

3.3.2 Support Vector Machines

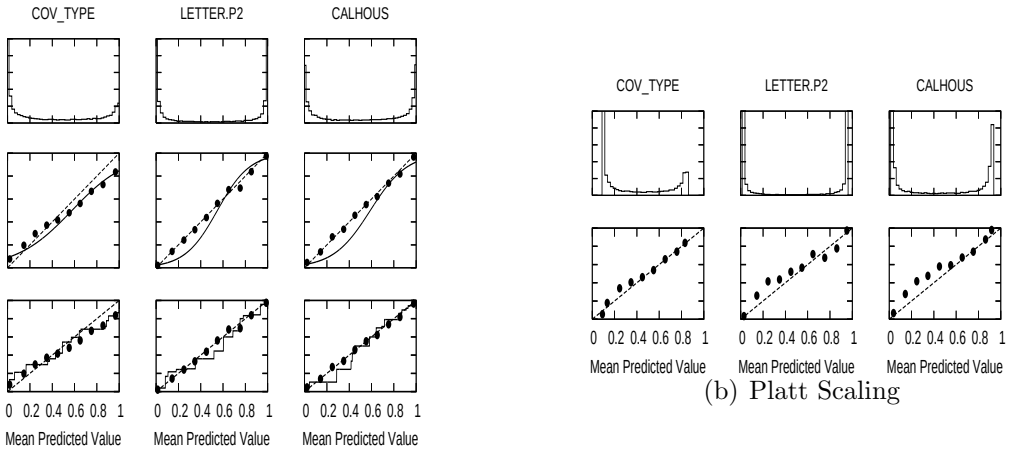
Another widely used maximum margin learning method is the Support Vector Machine (SVM) (Vapnik, 1998). Designed mainly for classification tasks, SVMs attempt to find a decision surface such that not only are the training examples classified correctly, but also there is a large separation (margin) between the predictions for examples in different classes. By explicitly trading off training accuracy for margin size, SVMs generate a regularized solution that provides protection against overfitting and ensures good generalization performance. In the process, however, the predictions generated by SVMs loose any probabilistic interpretation.

Since we are interested in assessing the calibration of the best SVM models we use a variety of kernel and parameter settings to find what works best for each problem. We train the models using SVMLight (Joachims, 1999) with the following

kernels: linear, polynomial of degree 2 and 3, radial with width 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1 and 2. For each kernel, we vary the regularization parameter by factors of ten from 10^{-7} to 10^{3}. To generate the histograms of predicted values and the reliability diagrams in this section we use, for each problem, the SVM model with the lowest squared error either before or after calibration. For consistency with the other learning methods SVM's predictions are linearly scaled to [0,1] by $(x - min)/(max - min)$. This scaling does not affect any of the results in this section.

Fortunately, there is a connection between the predictions of an SVM and the posterior class probabilities. Platt (1999) observed that for a number of real problems the reliability plots for SVM models have a sigmoidal shape. Figure 3.6(a) provides additional evidence that this behavior is common in real data sets. The figure shows histograms of predicted values and reliability diagrams for the COV_TYPE, COD and BACT problems. Graphs for all eleven problems are included in Figure 3.17(a) in Appendix 3.A. With two exceptions, BACT and MG, a sigmoid provides a good fit for the reliability diagrams of the best SVM models. Also, as in the case of boosted decision trees and boosted decision stumps, the sigmoidal shape of the reliability plots co-occurs with the concentration of mass in the center of the histograms of predicted values. Calibrating the SVM models with either Platt Scaling or Isotonic Regression removes the shift in predicted values and generates well calibrated models with the reliability diagram close to the diagonal line. (See Figure 3.6(b).)

It is interesting to note that both SVMs and AdaBoost are max-margin or quasi-max-margin methods, and that for both methods the predictions for an example is proportional to the distance between the example and the decision surface in some induced feature space (both are linear classifiers in that space). An interesting open



(a) Uncalibrated

(b) Platt Scaling

Figure 3.7: Histograms of predicted values and reliability diagrams for neural networks before and after calibration.

question is whether the the sigmoidal reliability plots and the shift in predicted values are characteristic of all max-margin linear (in some feature space) classifiers, and if there is a theoretical justification for this behavior.

3.3.3 Artificial Neural Networks and Logistic Regression

Logistic Regression and Artificial Neural Nets (ANNs) have been widely used to make probabilistic predictions. We train neural nets with 1, 2, 4, 8, 16, 32 and 128 hidden units using gradient descent backprop with momentums of 0, 0.2, 0.5 and 0.9. For Logistic Regression we train both unregularized and regularized models varying the ridge (regularization) parameter by factors of 10 from 10^{-8} to 10^{4}.

Figure 3.7(a) shows histograms of predicted values and reliability plots for neural nets for COV_TYPE, LETTER.P2 and CALHOUS. Figures for the all problems are included in Figure 3.18(a) in Appendix 3.A. The reliability plots closely follow the diagonal line indicating that neural nets are well calibrated to begin with and do not need post-training calibration. (We are using early stopping to select the number of passes of backpropagation that yield the best squared

error on the calibration sets. If the neural nets are trained far past these stopping points sigmoidal shaped reliability diagrams emerge for some problems. Only the COV_TYPE problem appears to benefit a little from post-training calibration. On the other problems both calibration methods appear to be striving to approximate the diagonal line, a task that isn't natural to either of them. Because of this, scaling might hurt neural net calibration a little. The sigmoids trained with Platt's method have trouble fitting the tails properly, effectively pushing predictions away from 0 and 1 as can be seen in the histograms in Figure 3.7(b). This shift of predicted values away from 0 and 1, however, is qualitatively different form the shift in predictions displayed by uncalibrated boosted trees and stumps and uncalibrated SVMs. Logistic Regression behaves similarly to neural nets: it also is very well calibrated to start with and post-training calibration seems to hurt because both methods have a difficult time fitting the diagonal line (see Figure 3.19).

The histograms for uncalibrated neural nets in Figure 3.18(a) look similar to the histograms for boosted trees *after* Platt Scaling in Figure 3.2(b), giving us confidence that the histograms reflect the underlying structure of the problems. For example, we could conclude that the CALHOUS, COV_TYPE, LETTER.P1, LETTER.P2 and HS problems, *given the available features*, have well defined classes with a small number of cases in the "gray" region, while in the SLAC and COD problems the two cases have high overlap with significant uncertainty for most cases. For the MEDIS problem it seems that there are many cases that are negative with high probability, but for cases labeled positive there is never a high probability that a case is positive. For the BACT problem on the other hand, it seems that there is more confidence about the positive cases than the negative ones.

It is interesting to note that neural networks with a single sigmoid output

unit can be viewed as a linear classifier (in the span of it's hidden units) with a sigmoid at the output that calibrates the predictions. In this respect neural nets are similar to SVMs and boosted trees after they have been calibrated using Platt's method. Another important observation is that while neural nets do not need the extra calibration set for post training calibration, they need it for early stopping, which is critical for good calibration. In an extreme case, if no early stopping were to be performed (the nets are trained to completion), and if the hidden layer is large enough, the neural net models could fit the training data perfectly and only predict zero or one and consequently have bad calibration. Similarly to boosted decision stumps calibrated with Logistic Correction (or trained to optimize log-loss), Logistic Regression does not need extra independent data for early stopping, but it is not expressive enough to capture the complexities of many real data sets.

3.3.4 Decision Trees

A decision tree partitions the instance space into axes-parallel hyper-rectangular regions corresponding to the leafs of the tree. In the binary classification case, for all the cases falling into one of these regions, the tree makes a prediction equal to the positive class frequency in the training set at the respective leaf. So the predictions made by a decision tree can be seen as an estimate of the average conditional class probability of all the cases in one such hyper-rectangle. Pruning is usually used to make sure each leaf has enough training cases to make this estimate reliable. The downside however, is that a pruned tree will have a small number of leafs (each corresponding to larger hyper-rectangles) especially if the training set is small, and will only be able to make coarse predictions, hurting their calibration. In the extreme case, when the tree has only one leaf, it will predict for any example the fraction of positive cases in the training set. While

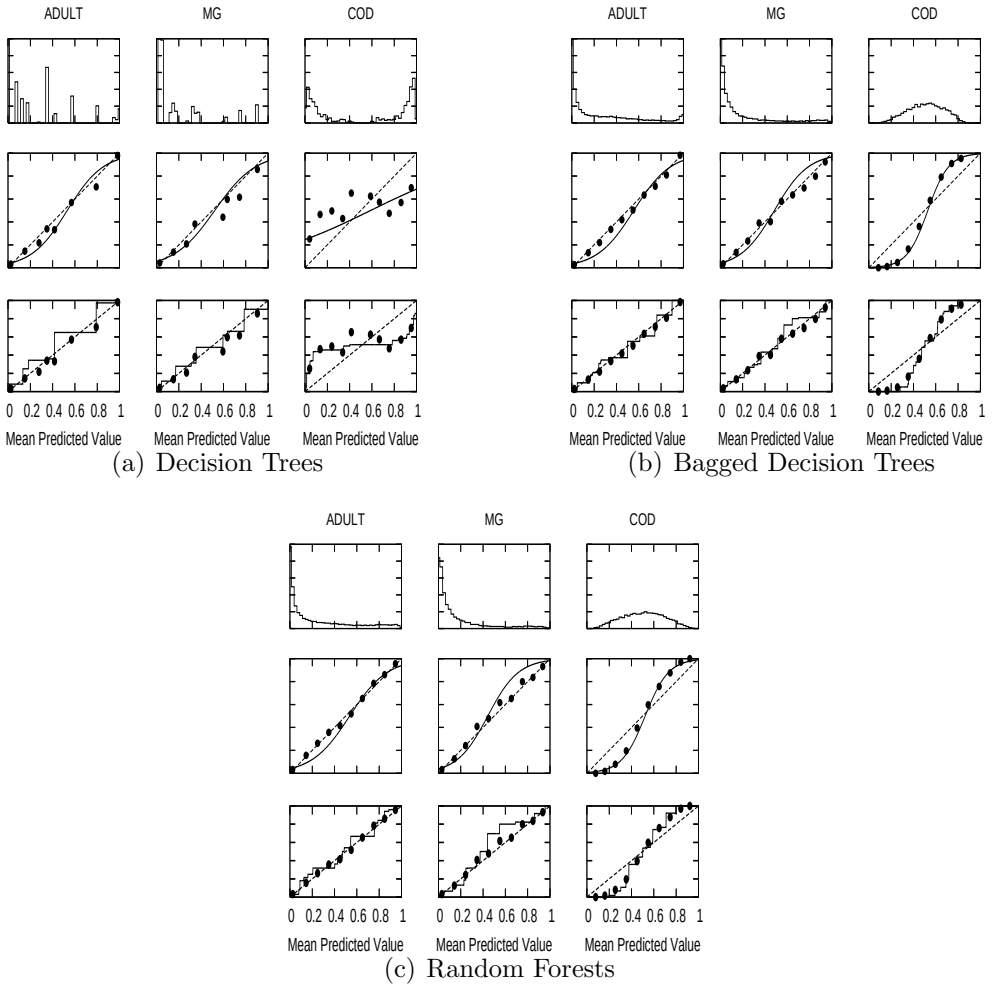


Figure 3.8: Histograms of predicted values and reliability diagrams for (a)Decision Trees, (b)Bagged Decision Trees and (c)Random Forests

this is a very reliable estimate of the average conditional class probabilities of the data set, it is not very useful.

Reducing or even eliminating pruning will generate trees with a larger number of leafs, consequently allowing the tree to make predictions that are more fine-grained. Reducing pruning however, increases the variance of the decision trees, hurting the calibration of their predictions. When the decision tree has a large number of leafs, only a few training cases end up in each one, making the probabilistic estimates unreliable. Also a large tree is likely to overfit the training

data, generating overly optimistic predictions (i.e. the tree will overestimate, or underestimate, the class conditional probabilities). To alleviate this problem, the predictions at each leaf are corrected using a smoothing method such as Laplacian or Bayesian smoothing (Buntine, 1992).

We train decision trees using varying splitting criteria, pruning options and smoothing methods. We use all of the tree types in Buntine's IND package (Buntine & Caruana, 1991): BAYES, ID3, CART, CART0, C4, MML, and SMML. We also generate trees of type C44LS (C4 with no pruning and Laplacian smoothing), C44BS (C44 with Bayesian smoothing), and MMLLS (MML with Laplacian smoothing). See (Provost & Domingos, 2003) for a description of C44LS.

The prediction histograms and reliability diagrams for the decision tree models with the best squared error (either before or after calibration, whichever has better squared error) are shown in Figure 3.8(a) for the ADULT, MG and COD problems and in Figure 3.20(a) in Appendix 3.A for all eleven problems. The graphs show both issues discussed above. On some problems such as ADULT, MG and SLAC the histograms show that the trees make coarse predictions and a large number of cases receive the same predicted values. The reliability diagrams in such cases are relatively close to the diagonal indicating the predictions are reliable. For other problems such as COD, HS, and COV_TYPE, the decision trees are able to predict a lot of different values, but the predictions suffer from high variance and overfitting. The overfitting is evident in the COD and HS problems, where the trees are overly confident about their predictions, and the reliability diagrams are far under and over the diagonal line when approaching one and zero respectively.

Neither calibration method is able to help with the coarseness problem, and Platt Scaling might actually hurt because a sigmoid is not the right function to use in these cases. With the overfitting problem, both calibration methods might help,

because they are able to adjust the predictions at each leaf using independent data that was not used to train the tree. Here, too, Isotonic Regression has an advantage since the overconfident predictions generate an inverted sigmoidal shaped reliability digram, a shape that is totally wrong for Platt Scaling, but that Isotonic Regression is able to fit.

3.3.5 Bagged Decision Trees and Random Forests

Bagged Decision Trees (Breiman, 1996) and Random Forests (Breiman, 2001) address the two shortcomings of decision trees described in the previous section. Both learning algorithms were designed to reduce the variance of the decision trees models thus addressing overfitting. Also, both methods average the predictions of multiple different decision trees, so they can make more fine grained predictions even if the constituent trees have few leafs.

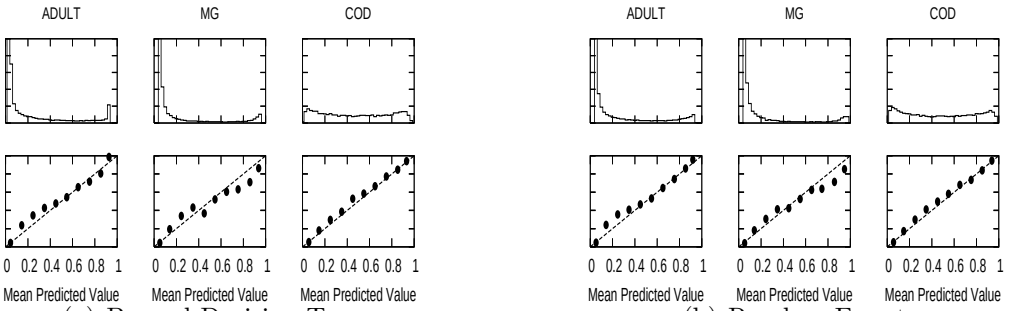
For each tree type listed above, we create a bagged trees model of 100 trees. To train Random Forests models we used the Breiman-Cutler FORTRAN implementation. Since the trees in a random forest have more variance than the ones in a bagged ensemble, we create random forests of 1024 trees. The size of the feature set considered at each split is 1, 2, 4, 6, 8, 12, 16 or 20, a wider range than Breiman and Cutler suggest is needed.

Examining the histograms and reliability diagrams for bagged trees and random forests in Figures 3.8(b) and 3.8(c) (and in Figures 3.21(a) and 3.22(a) in Appendix 3.A) two distinct behaviors can be identified. On some problems such as ADULT, MG, SLAC, and CALHOUS, both bagged trees and random forest make fine grained, well calibrated predictions. Given that bagged trees and random forests are well calibrated on these problems, we can deduce that regular decision trees also are well calibrated on average, in the sense that if the decision

trees are trained on different samples of the data and their predictions averaged, the average will be well calibrated. If instead of decision trees, the base level models for bagging would have been poorly calibrated models such as SVMs or boosted trees that present a systematic distortion of the probabilities, the bagged models would have also been poorly calibrated. There is nothing bagging can do to fix systematic distortions, it only takes care of distortions generated by variance.

Interestingly, on other problems, such as COD, LETTER.P2, and HS, bagged trees and random forests seem to exhibit, although to a lesser extent, the same behavior as the max-margin methods: predicted values are slightly pushed toward the middle of the histogram and the reliability plots show a sigmoidal shape. While further investigations are needed to identify the reason(s) for this behavior, there is one phenomenon that might provide at least a partial explanation. Methods such as bagging and random forests that average predictions from a base set of models can have difficulty making predictions near 0 and 1 because variance in the underlying base models will bias predictions that should be very near zero or very near one away from these values. Because predictions are restricted to the interval [0,1], errors caused by variance tend to be one-sided near zero and one. For example, if a model should predict $p = 0$ for a case, the only way bagging can achieve this is if *all* bagged trees predict zero. If we add variance to the trees that bagging is averaging over, this will cause some trees to predict values larger than 0 for this case, thus moving the average prediction of the bagged ensemble away from 0. We observe this effect more strongly with random forests because the base-level trees trained with random forests have relatively high variance because of feature subsetting.

As in the case of neural nets and logistic regression, post-training calibration of bagged trees and random forests models might actually hurt on the problems where



(a) Bagged Decision Trees (b) Random Forests

Figure 3.9: Histograms of predicted values and reliability diagrams after calibration with Platt Scaling for (a)Bagged Decision Trees and (b)Random Forests.

the models are already well calibrated. The prediction histograms in Figure 3.9 show that, on these problems, Platt Scaling pushes the predicted values away from zero and one because of its inability of fit well near zero and one. On the other hand, on the problems where bagged trees and random forests are not well calibrated, post-training calibration improves the quality of the predictions, with Platt Scaling being an excellent fit given the sigmoidal shape of the reliability diagrams.

3.3.6 Memory Based Learning

Memory based learning algorithms come in many different flavors that differ in how distances are computed and how each nearest neighbor's vote is weighted. We experiment with Euclidean distance with all attributes scaled to zero mean and standard deviation one, and with Euclidean distance with attributes weighted by their gain ratio. We train vanilla K-nearest neighbor models where each neighbor receives equal weight, distance weighted KNN where each neighbor is weighted inversely proportional to it's distance, and locally weighted averaging where the weight falls of exponentially with the distance. We use 26 values of K ranging from $K = 1$ to $K = |trainset|$ with smaller values more densely sampled, and with

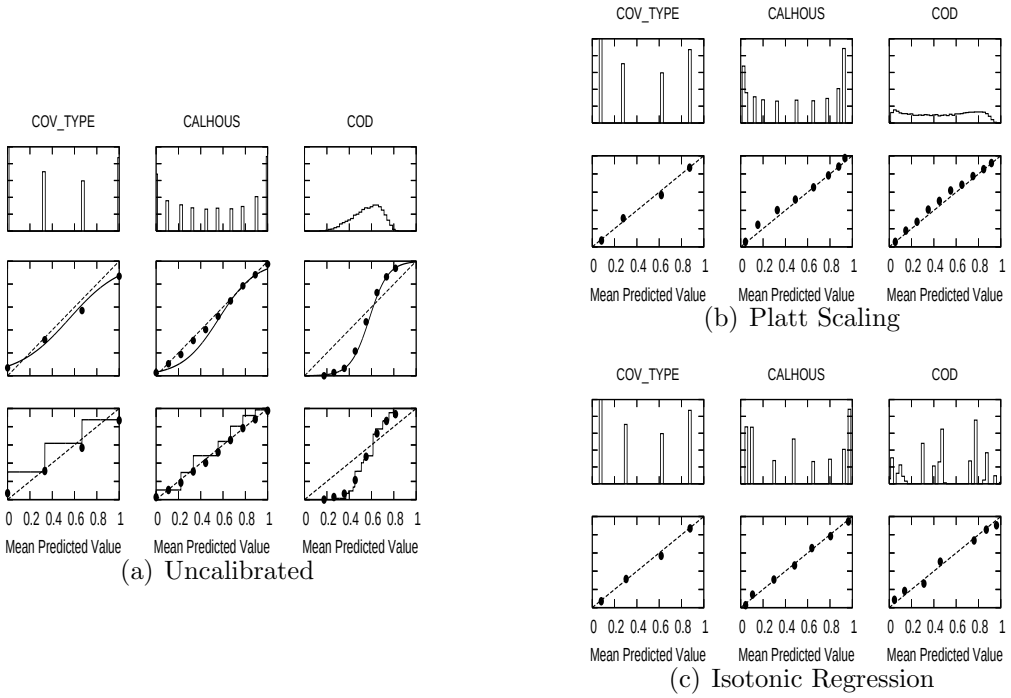


Figure 3.10: Histograms of predicted values and reliability diagrams for memory based learning before and after calibration.

kernel widths varying from 2^0 to 2^{10} times the minimum distance between any two points in the train set for locally weighted averaging.

The predictions made by memory based leaning algorithms can be interpreted as estimates of the conditional class probability. The estimate is generated by allowing each training case in the neighborhood to cast a (possibly weighted) vote for it's class. If the size of the neighborhood is too small (e.g. small K), then the estimates will not be very reliable, and depending on the particular variant used the predictions might be very coarse. An example of this situation is depicted in Figure 3.10(a) for the COV_TYPE problem, where the best memory based learning method is a vanilla KNN with $K = 3$ and attributes are weighted by gain ratio. The reliability diagram shows that the predictions are overconfident, especially in the two tails, indicating overfitting. As in the case of decision trees, calibration can not help with the coarseness, but it might help with overfitting.

(See Figures 3.10(b) and 3.10(c)). On the CALHOUS problem, where the optimal K is 9, the reliability diagram is close to the diagonal line, but the predictions are still too coarse (only ten distinct values). In both these cases Isotonic Regression seems to work sightly better than Platt Scaling because a sigmoid is not the right shape for calibrating these predictions.

When the neighborhood is large (e.g. the optimal K is large), as is the case for the COD and SLAC problems, the predictions are shifted toward the middle of the plot, and the reliability diagrams display a sigmoidal shape. This might be also due to an edge effect; because the vote of each neighbor is either zero or one, in order for the model to make a prediction of one, all the neighbors must be positive cases, which is unlikely when K is large. Calibration is largely able to fix this problem, and in particular Platt Scaling is well suited in this case.

3.3.7 Naive Bayes

Naive Bayes is well known to have a bias toward predicting extreme values close to zero and one. This bias is due to the unrealistic assumption that the attributes are independent given the class. This assumption causes Naive Bayes to over count the evidence and make overconfident predictions.

This bias is demonstrated by the prediction histograms and reliability diagrams for the ADULT and MEDIS problems in Figure 3.11(a), and it is also present to a lesser extent in the LETTER.P2 problem. The reliability diagrams for ADULT and MEDIS are far below the diagonal line when the predicted values approach one (right side of the diagram), indicating that the Naive Bayes models predict the positive class with too much confidence. The bias is especially noticeable when comparing the histograms of predicted values for Naive Bayes with histograms of predicted values for well calibrated models such as neural nets or boosted trees that

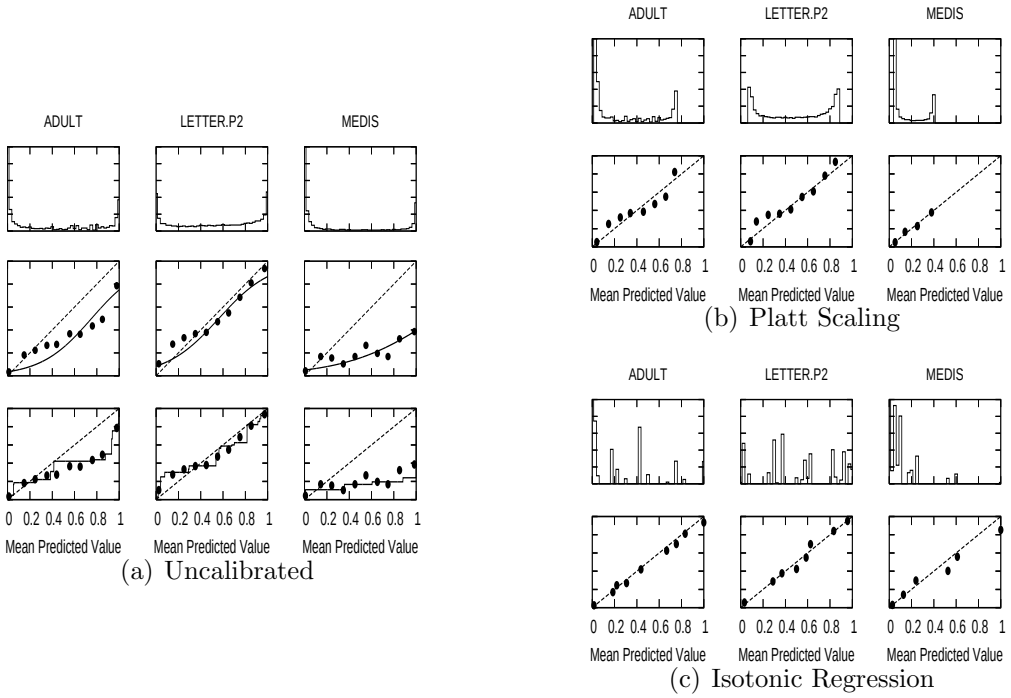


Figure 3.11: Histograms of predicted values and reliability diagrams for Naive Bayes before and after calibration.

have been calibrated with Platt Scaling. The well calibrated models never predict a value higher than 0.9 for the MEDIS problem, and make very few predictions close to one for the ADULT problem. Naive Bayes, however, makes many predictions close to one on both of these problems.

This push of predicted values toward zero and one generates reliability diagrams that have a roughly inverted sigmoidal shape. While Platt Scaling is still helping to improve calibration, it is clear that a sigmoid is not the right transformation to calibrate Naive Bayes models. The histograms in figure 3.11(b) show that Platt Scaling pushes the predicted values away from zero and one by essentially squashing them. This squashing generates prediction histograms that are quite different from the histograms for well calibrated models. For example, on ADULT and MEDIS, histograms of predicted values for neural nets and boosted trees calibrated with Platt Scaling show an almost monotonic decay as the predicted values approach

one, while the histograms in Figure 3.11(b) have a sudden increase at the left end.

Isotonic Regression is able to fit better the inverted sigmoidal shape of the reliability diagrams, and for most problems the reliability diagrams for Naive Bayes models calibrated with Isotonic regression are closer to the diagonal line than the reliability diagrams for Naive Bayes calibrated with Platt Scaling (See Figure 3.11 and Figure 3.24). So Isotonic Regression seems to be a better choice for calibrating Naive Bayes models.

3.4 Quantitative Analysis of Performance

Our goal in this section is to use squared error and cross-entropy performance as a quantitative measure of the calibration of a model and conduct an empirical comparison across the learning algorithms and calibration methods discussed earlier in the chapter. Rather than delving into details of performances on each individual problem, we will maintain a more high-level view and only present and discuss average performance across the eleven test problems.

For each learning algorithm, we use all the the parameter settings described in the previous section to train classifiers, and calibrate each classifier with Isotonic Regression and Platt Scaling. In the case of boosted trees and stumps we also calibrate the models using Logistic Correction and train models that directly optimize log-loss. Models are trained on 4000 points and calibrated on 1000 independent points (if needed). For each data set, learning algorithm, and calibration method, we select the model with the best performance using the same 1000 points used for calibration, and report it's performance on the large final test set. We use five fold cross-validation to obtain five trials.

Figure 3.12 shows the root mean squared error (left) and mean cross-entropy (right) for each learning method before and after calibration. Each bar averages

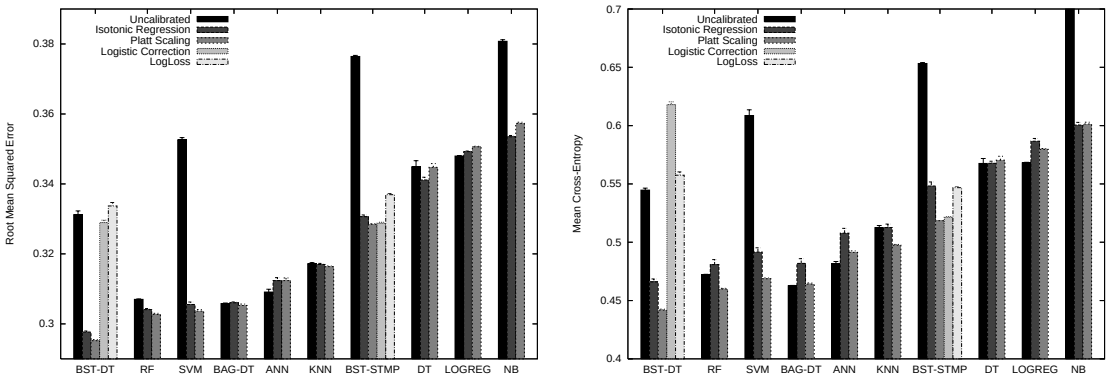


Figure 3.12: Performance of learning algorithms

over five trials on each of the eleven problems. Error bars representing one standard error are shown, but they are too small to be seen in some cases. The learning algorithms are roughly ordered by their squared error performance, with the best algorithm on the left. Boosted decision trees calibrated with Platt Scaling are clearly the best method with significantly lower squared error and cross-entropy when compared to the other learning algorithms. Boosted trees are followed by calibrated random forests, calibrated SVMs, bagged decision trees and uncalibrated neural nets. If we only look at the uncalibrated models however, bagged trees, random forests and neural nets are the best models. At the other end of the spectrum, boosted stumps, decision trees, logistic regression and Naive Bayes have the worst performance.[2] The main reason why boosted stumps, logistic regression and Naive Bayes perform poorly regardless of the calibration method used is not that these models could not be calibrated, but because the models learned using these algorithms are too simple to capture the full complexity of most datasets. Logistic regression and Naive Bayes learn linear models, while boosted stumps are unable to represent any interactions between attributes (Friedman et al., 2000). Similarly the performance of decision trees suffers because they have very high

[2]The mean cross-entropy for the uncalibrated Naive Bayes models is actually 1.06, but the figure is cut at 0.7 to better show the interesting region.

variance or make coarse predictions, and there is little that calibration can do to fix these problems. This ranking of learning algorithms by their performance is fairly consistent whether we look at squared error or at cross-entropy. The only difference is that bagged trees have slightly better cross-entropy than SVMs calibrated with Platt's method, while having slightly worse squared error.

As expected, the probabilities predicted by four learning methods — boosted trees, SVMs, boosted stumps, and Naive Bayes — are dramatically improved by calibration, while neural nets and logistic regression models are actually hurt a little by post-training calibration. For random forests, bagged trees and memory based learning we have seen in the previous section that Platt Scaling helps on some problems that display sigmoidal reliability plots, but hurts on other problems where the models already are well calibrated. For random forest and memory based learning the benefit is greater than the loss, so on average the models calibrated with Platt Scaling perform better. For bagged trees it's a wash. Comparing the left and the right graphs in Figure 3.12 it looks as if Isotonic Regression does perform slightly worse than Platt Scaling for cross-entropy for all learning methods. When looking at squared error, Isotonic Regression performs a little better than Platt Scaling for decision trees and Naive Bayes.

We have seen in Sections 3.3.1 and 3.3.1 that boosted trees calibrated with Logistic Correction and boosted trees trained to optimize log-loss only output zero or one predictions for most problems, which explains their poor squared error and cross-entropy performance. On the other hand, for boosted stumps, Logistic Correction performs on par with Platt Scaling. This makes Logistic Correction very appealing, especially since it does not require any extra time or data for calibration. Directly optimizing log-loss with boosted stumps performs a few percent worse than Logistic Correction, which is consistent with the results reported

by Lebanon and Lafferty (2001). Unfortunately, the main reason why Logistic Correction works well with boosted stumps and not with boosted trees is that boosted stumps have limited expressive power. Most datasets are too complex to be learned well using boosted stumps. Boosted trees however, are able to learn models for these complex datasets well, and when calibrated with Platt Scaling produce excellent probabilistic predictions.[3]

3.5 Learning Curve Analysis

In this section we present a learning curve analysis of the two calibration methods, Platt Scaling and Isotonic Regression. The goal is to determine how effective these calibration methods are as the amount of data available for calibration varies. For this analysis, for each problem, learning algorithm, and calibration method, we select the model with the lowest squared-error among the models trained with each learning algorithm. We vary the size of the calibration set from 32 cases to 8192 cases by factors of two.

The plots in Figure 3.13 show the average squared error over the ten test problems (the HS problem is left out because there is not enough data available for calibration set sizes bigger than 4092 points). For each problem, we perform ten trials. Error bars representing one standard error are shown on the plots, but are so narrow that they may be difficult to see.

The nearly horizontal lines in the graphs show the squared error prior to calibration. These lines are not perfectly horizontal only because the test sets change as more data is moved into the calibration sets. Each plot shows the squared er-

[3]We have also run experiments with boosting 2-level decision trees instead of the 1-level stumps. Boosted 2-level trees did outperform boosted stumps, but did not perform as well as boosting full trees. Moreover, 2-level trees are complex enough that Logistic Correction and optimizing to log-loss are no longer as effective as Platt Scaling or Isotonic Regression.

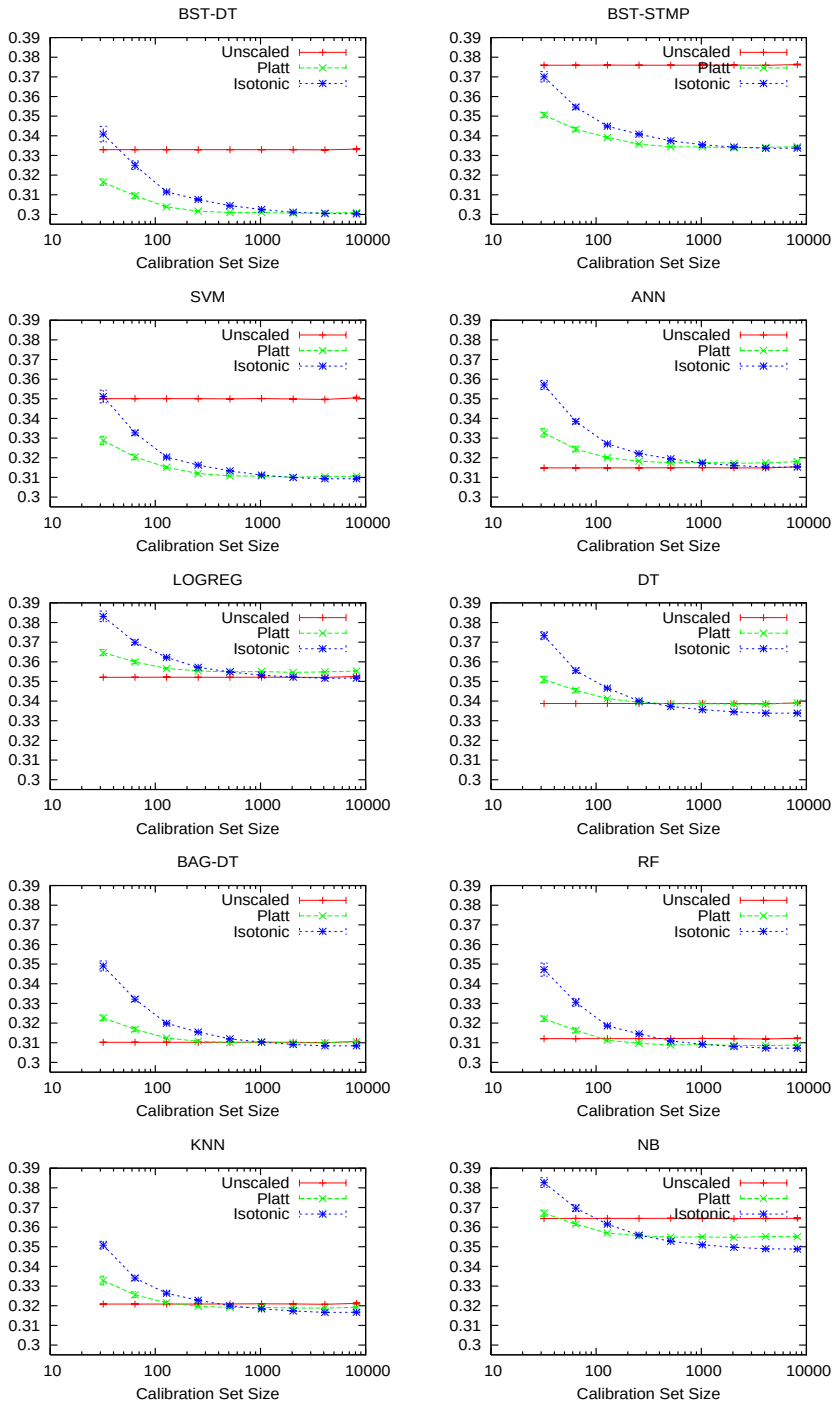


Figure 3.13: Learning Curves for Platt Scaling and Isotonic Regression (averages across 10 problems).

ror after calibration with Platt's method or Isotonic Regression as the size of the calibration set varies from small to large. When the calibration set is small (less than about 200-1000 cases), Platt Scaling outperforms Isotonic Regression with all nine learning methods. This happens because Isotonic Regression is more powerful than Platt Scaling, so it is easier for it to overfit when the calibration set is small. Platt's method on the other hand, besides being more constrained, also has some overfitting control built into it (see Section 3.2.1). Also, with smaller calibration sets, Isotonic Regression will make coarser predictions because the piecewise constant function it fits will have larger plateaus.

As the size of the calibration set increases, the learning curves for Platt Scaling and Isotonic Regression join, or even cross. When there are 1000 or more points in the calibration set, Isotonic Regression always yields performance as good as, or better than, Platt Scaling.

For learning methods that make well calibrated predictions such as neural nets, bagged trees, and logistic regression, neither Platt Scaling nor Isotonic Regression yields much improvement in performance even when the calibration set is very large. With these methods calibration is not beneficial, and actually hurts performance when the the calibration sets are small.

For the max-margin methods, boosted trees, boosted stumps and SVMs, calibration provides an improvement even when the calibration set is small. In Section 3.3 we saw that a sigmoid is a good match for boosted trees, boosted stumps, and SVMs. As expected, for these methods Platt Scaling performs better than Isotonic Regression for small to medium sized calibration (less than 1000 cases), and is virtually indistinguishable for larger calibration sets.

As expected, calibration improves the performance of Naive Bayes models for almost all calibration set sizes, with Isotonic Regression outperforming Platt Scal-

ing when there is more data. For the rest of the models: KNN, RF and DT post-calibration helps once the calibration sets are large enough.

3.6 Conclusions

In this chapter we examined the probabilities predicted by ten different classes of supervised learning algorithms. Maximum margin methods such as boosting and SVMs yield characteristic distortions in their predictions that are sigmoidal shaped. An empirical analysis indicates that although max-margin methods are effective at discriminating between cases with different likelihoods, their predictions are distorted by their willingness to trade-off reduced margin for "easier" cases that should have predicted values near zero or one, for increased margin for harder cases near the decision surface. Naive Bayes makes predictions with the opposite anti-sigmoid distortion. Methods such as neural nets and bagged trees predict well-calibrated probabilities. We examined the effectiveness of Platt Scaling and Isotonic Regression for calibrating the predictions made by different learning methods. Platt Scaling is most effective when the data is small, but Isotonic Regression is more powerful when there is sufficient data to prevent overfitting. After calibration, the models that predict the best probabilities are boosted trees, random forests, SVMs, uncalibrated bagged trees and uncalibrated neural nets.

Acknowledgment

This is joint work with Rich Caruana. Thanks to Bianca Zadrozny and Charles Elkan for the Isotonic Regression code, C. Young et al. at Stanford Linear Accelerator for the SLAC data, and Tony Gualtieri at Goddard Space Center for help with the HS data. This work was supported by NSF Award 0412930.

BIBLIOGRAPHY

Ayer, M., Brunk, H., Ewing, G., Reid, W., & Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, *5*, 641–647.

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

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, *2*, 63–73.

Buntine, W., & Caruana, R. (1991). *Introduction to IND and recursive partitioning* (Technical Report FIA-91-28). NASA Ames Research Center.

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proc. 23rd International Conference on Machine Learning (ICML'06)*.

Collins, M., Schapire, R. E., & Singer, Y. (2002). Logistic regression, adaboost and bregman distances. *Machine Learning*, *48*.

DeGroot, M., & Fienberg, S. (1982). The comparison and evaluation of forecasters. *Statistician*, *32*, 12–22.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, *38*.

Joachims, T. (1999). Making large-scale SVM learning practical. *Advances in Kernel Methods*.

Lebanon, G., & Lafferty, J. (2001). Boosting and maximum likelihood for exponential models. *Advances in Neural Information Processing Systems*.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. *Adv. in Large Margin Classifiers*.

Provost, F., & Domingos, P. (2003). Tree induction for probability-based rankings. *Machine Learning*.

Robertson, T., Wright, F., & Dykstra, R. (1988). *Order restricted statistical inference*. New York: John Wiley and Sons.

Rosset, S., Zhu, J., & Hastie, T. (2004). Boosting as a regularized path to a maximum margin classifier. *J. Mach. Learn. Res.*, *5*.

Schapire, R. (2001). The boosting approach to machine learning: An overview. *In MSRI Workshop on Nonlinear Estimation and Classification*.

Schapire, R., Freund, Y., Bartlett, P., & Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, *26*, 1651–1686.

Vapnik, V. (1998). *Statistical learning theory*. New York: John Wiley and Sons.

Zadrozny, B., & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. *ICML*.

Zadrozny, B., & Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. *Knowledge Discovery and Data Mining (KDD'02)*.

APPENDIX

3.A Histograms of Predicted Values and Reliability Diagrams

This appendix shows the histograms of predicted values and the reliablility diagrams before and after calibration for all the learning methods on each problem.

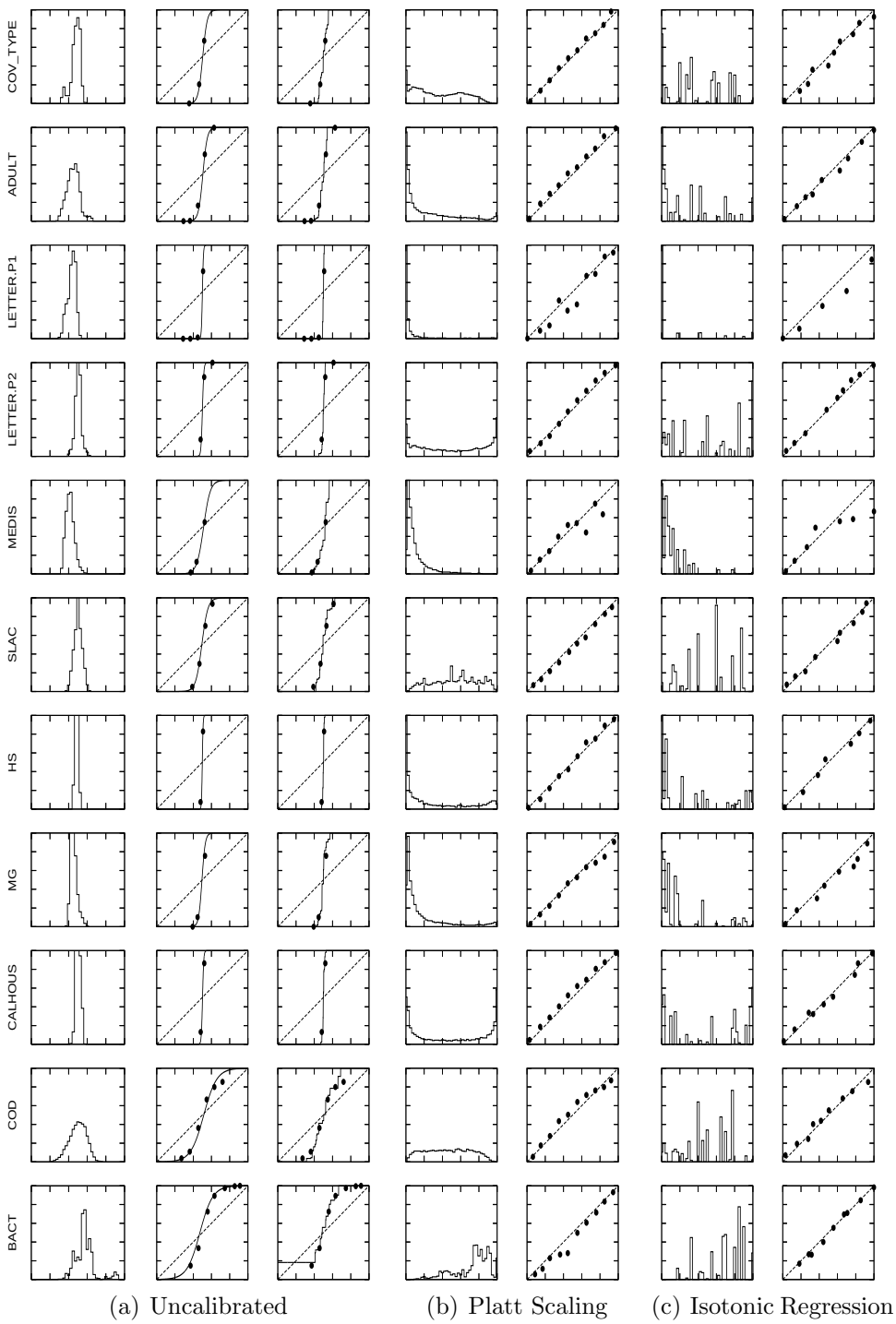
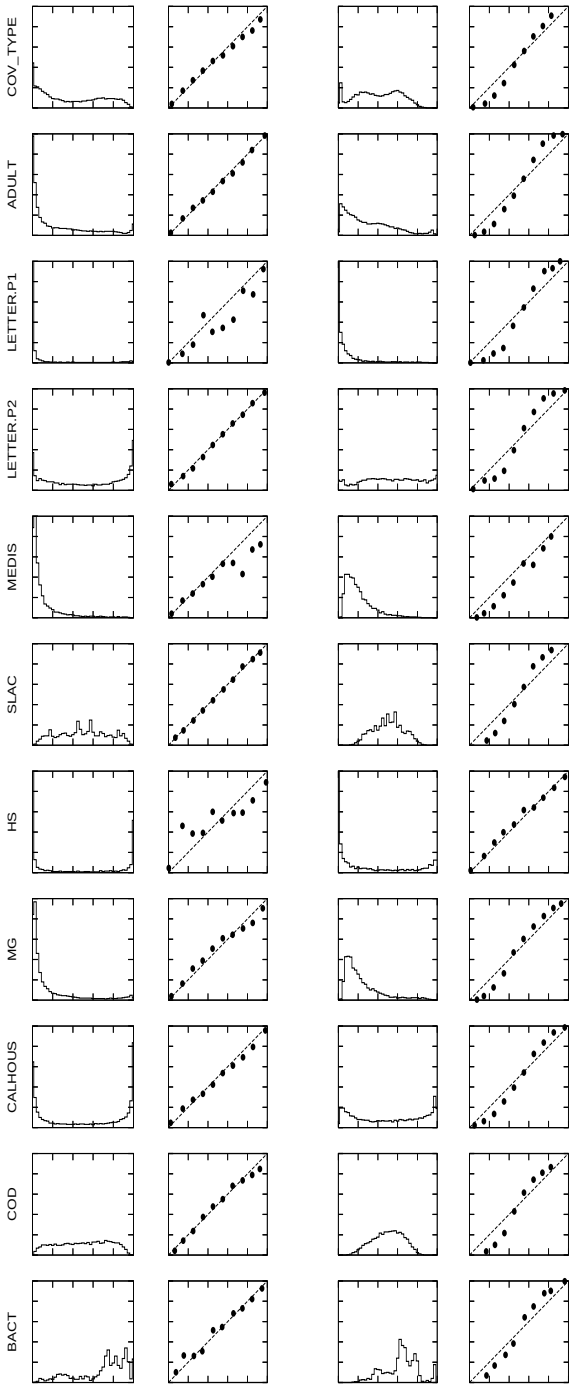
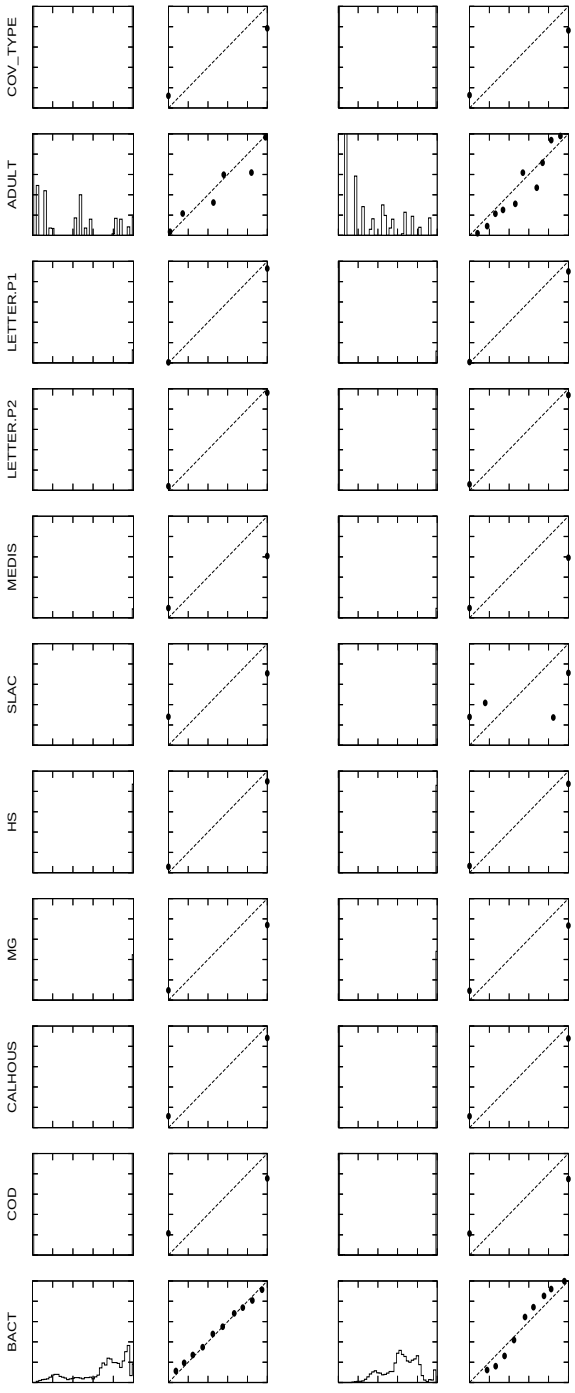


Figure 3.14: Histograms of predicted values and reliability diagrams for boosted decision stumps before and after calibration.



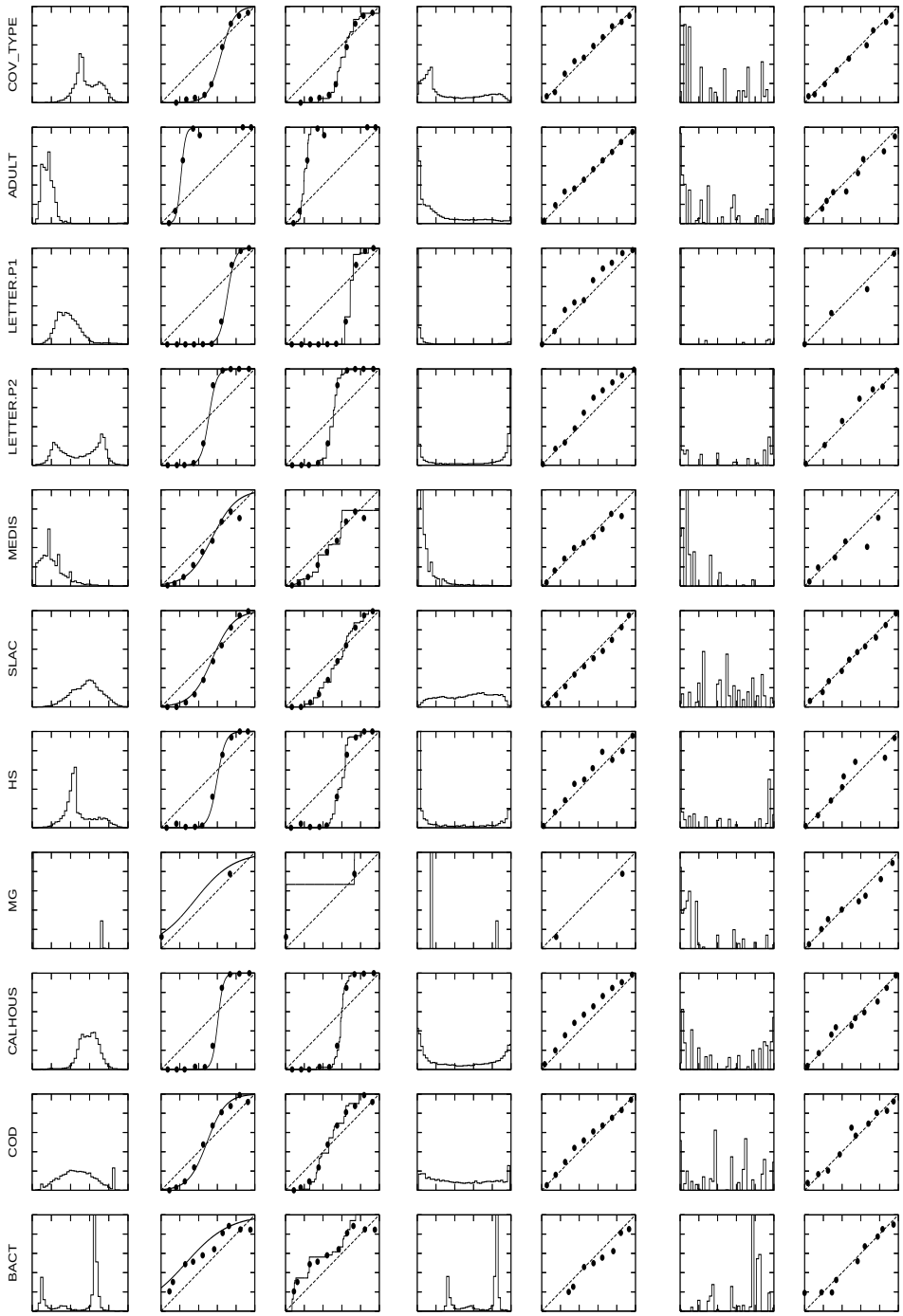
(a) Logistic Correction (b) Log-Loss

Figure 3.15: Histograms of predicted values and reliability diagrams for boosted decision stumps calibrated with Logistic Regression and for boosted decision stumps trained to optimize log-loss.



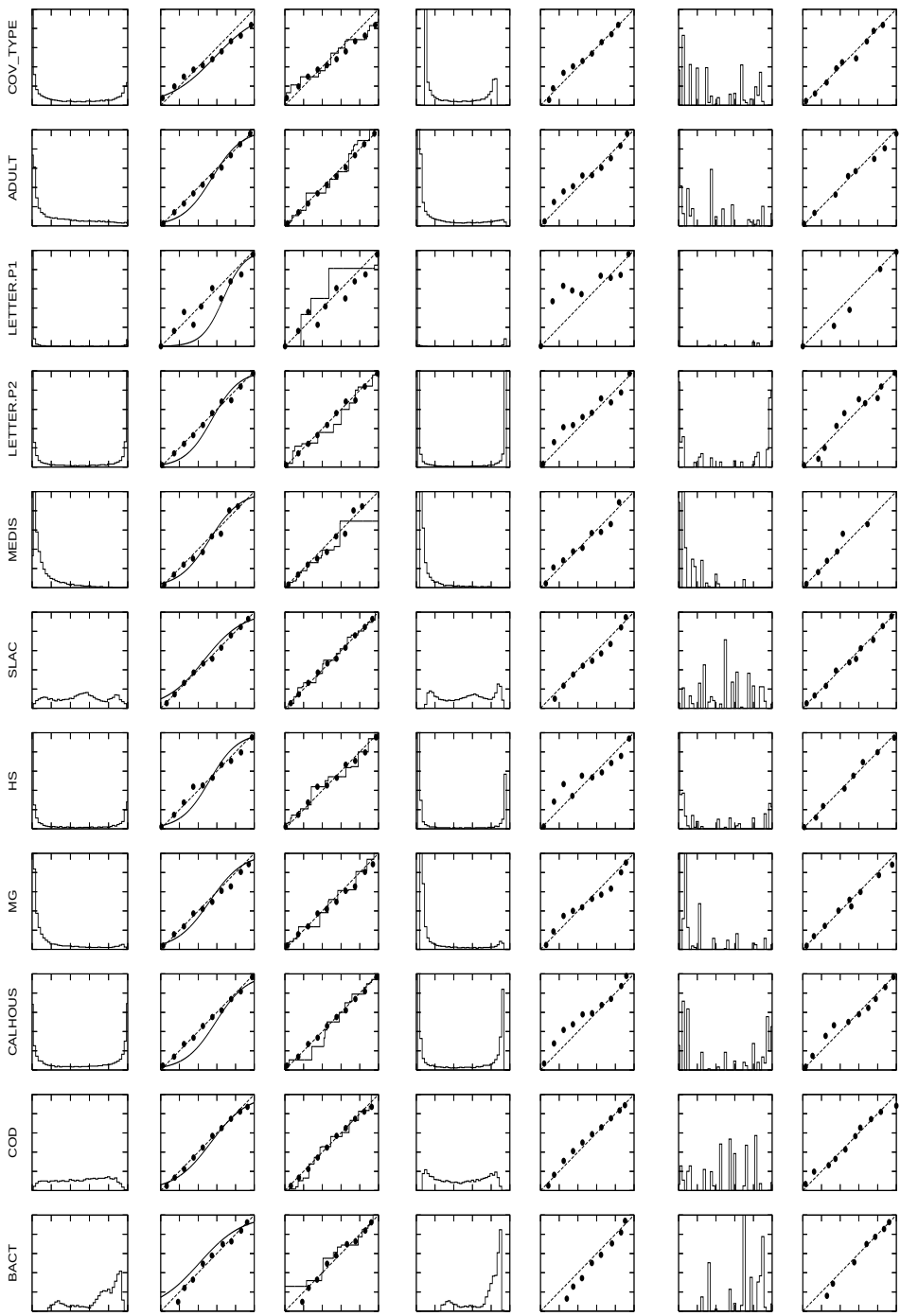
(a) Logistic Correction (b) Log-Loss

Figure 3.16: Histograms of predicted values and reliability diagrams for boosted decision trees calibrated with Logistic Regression and for boosted decision trees trained to optimize log-loss.



(a) Uncalibrated (b) Platt Scaling (c) Isotonic Regression

Figure 3.17: Histograms of predicted values and reliability diagrams for SVMs before and after calibration.



(a) Uncalibrated (b) Platt Scaling (c) Isotonic Regression

Figure 3.18: Histograms of predicted values and reliability diagrams for artificial neural networks before and after calibration.

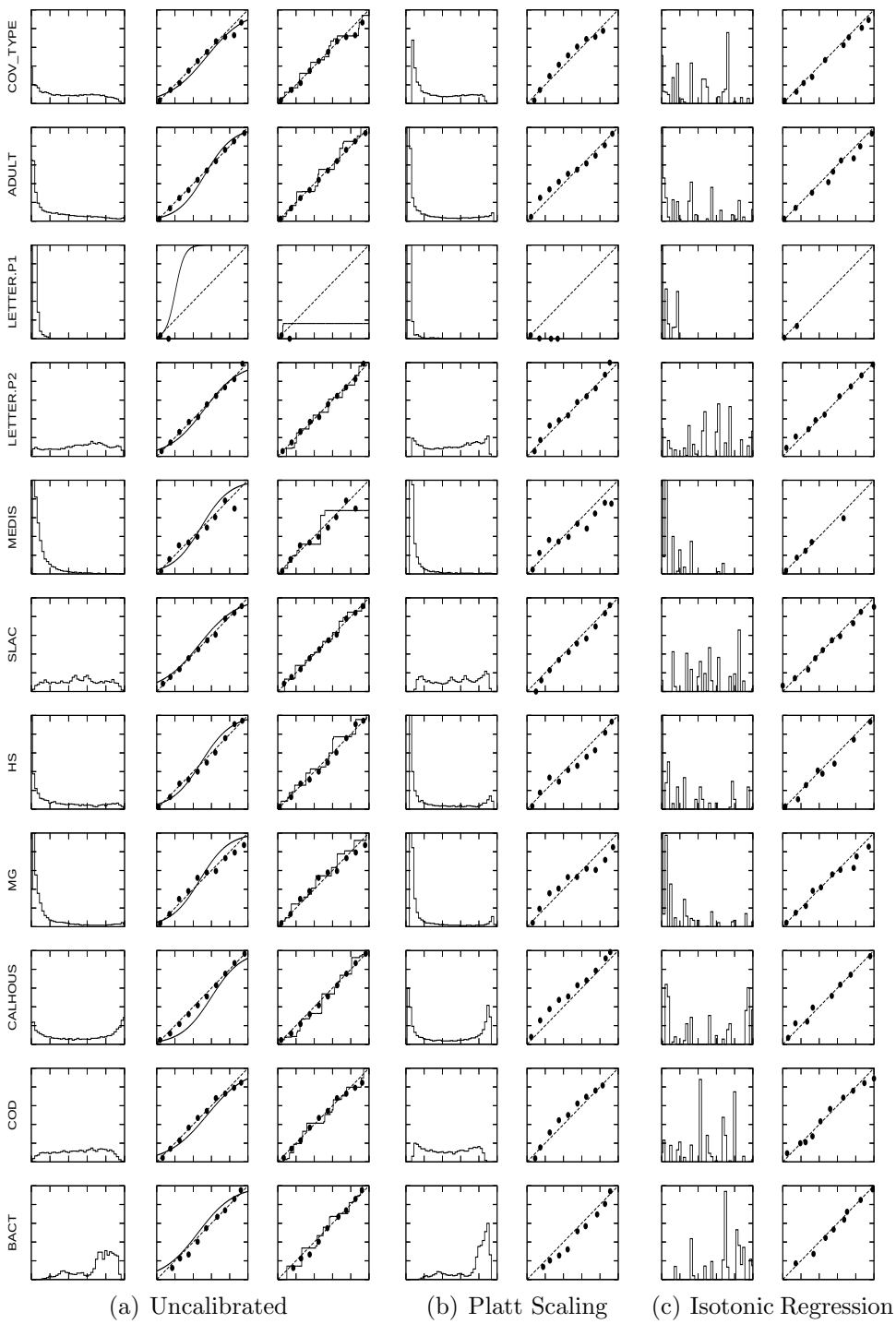
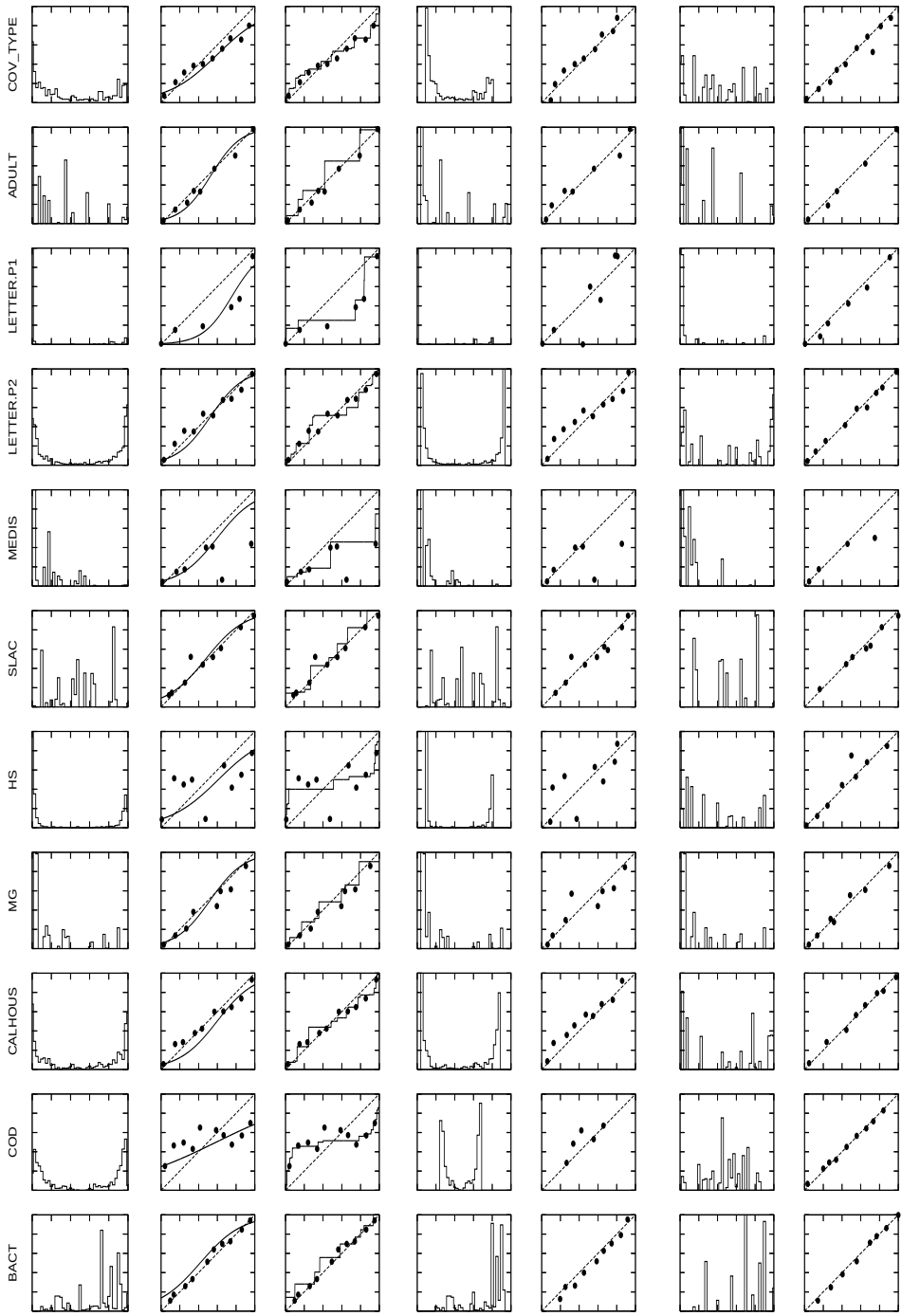


Figure 3.19: Histograms of predicted values and reliability diagrams for logistic regression before and after calibration.



(a) Uncalibrated (b) Platt Scaling (c) Isotonic Regression

Figure 3.20: Histograms of predicted values and reliability diagrams for decision trees before and after calibration.

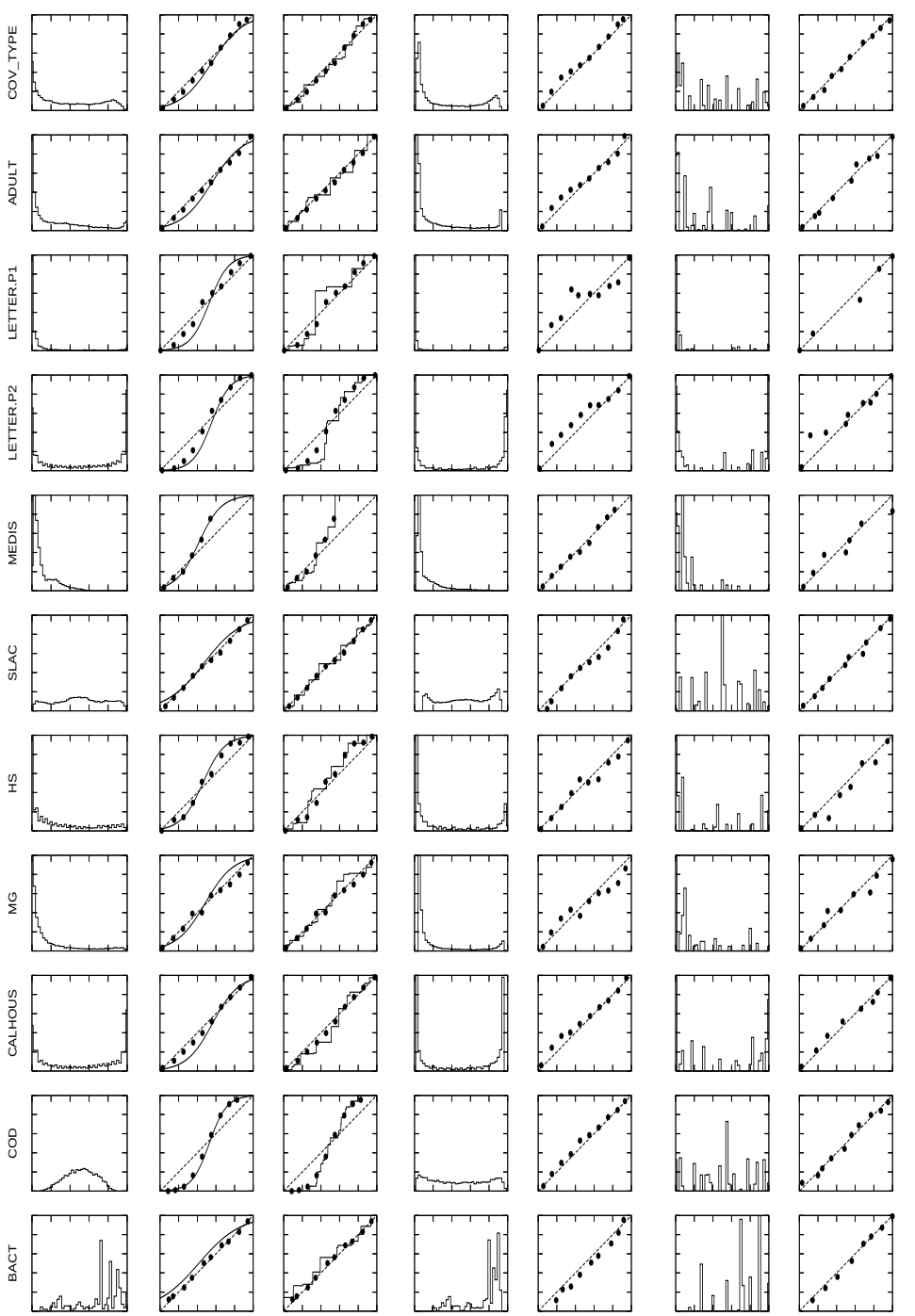


Figure 3.21: Histograms of predicted values and reliability diagrams for bagged decision trees before and after calibration.

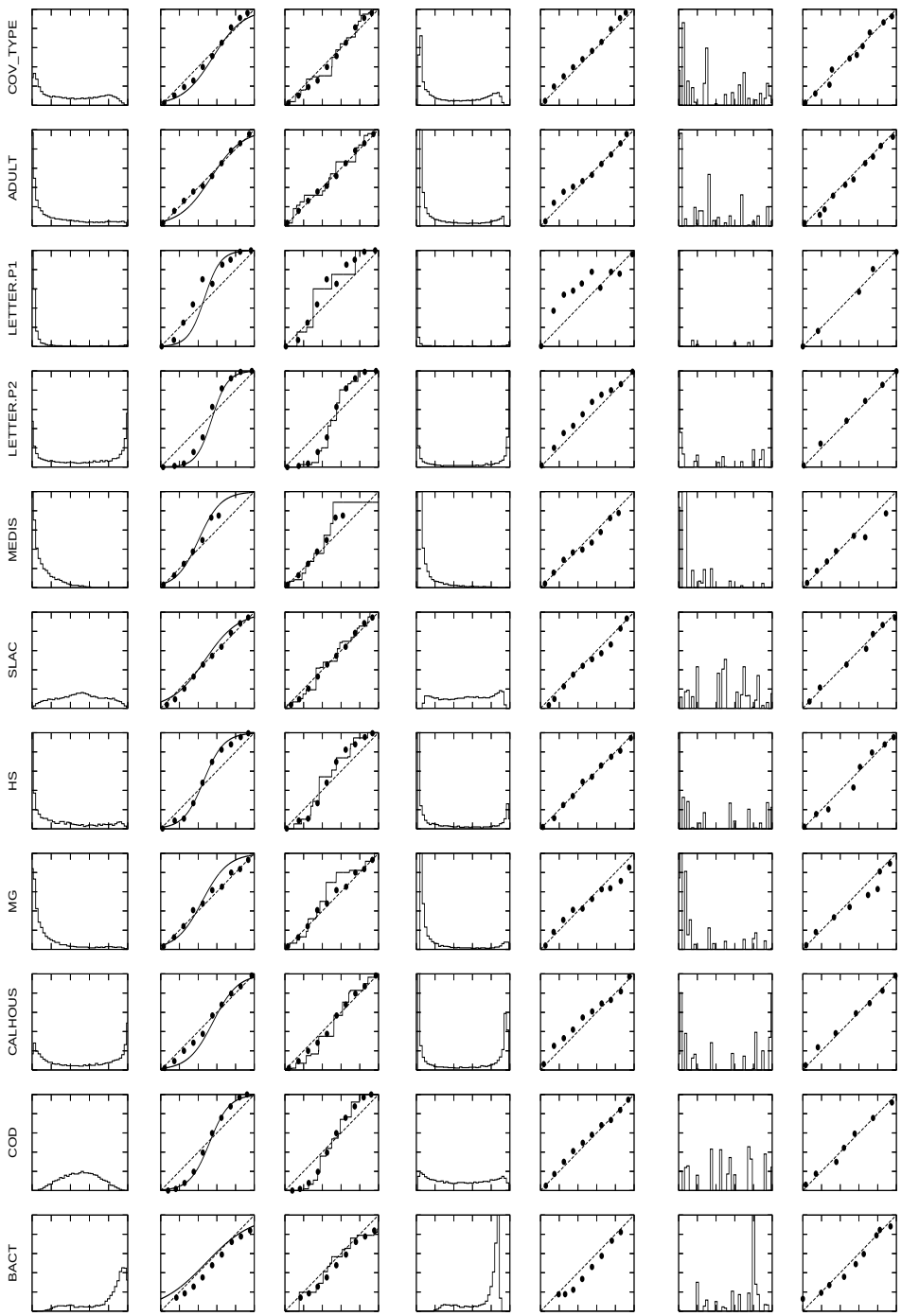
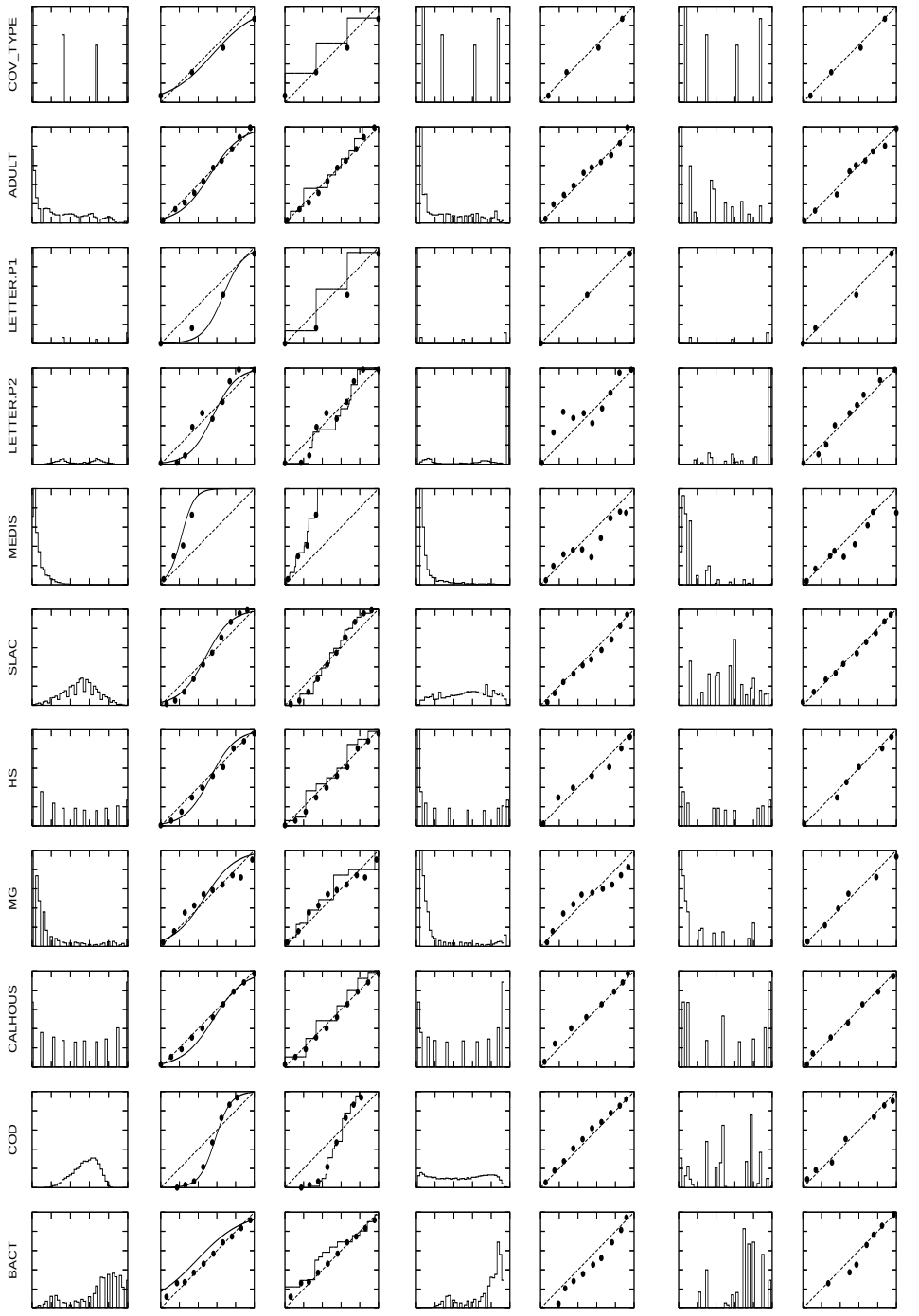
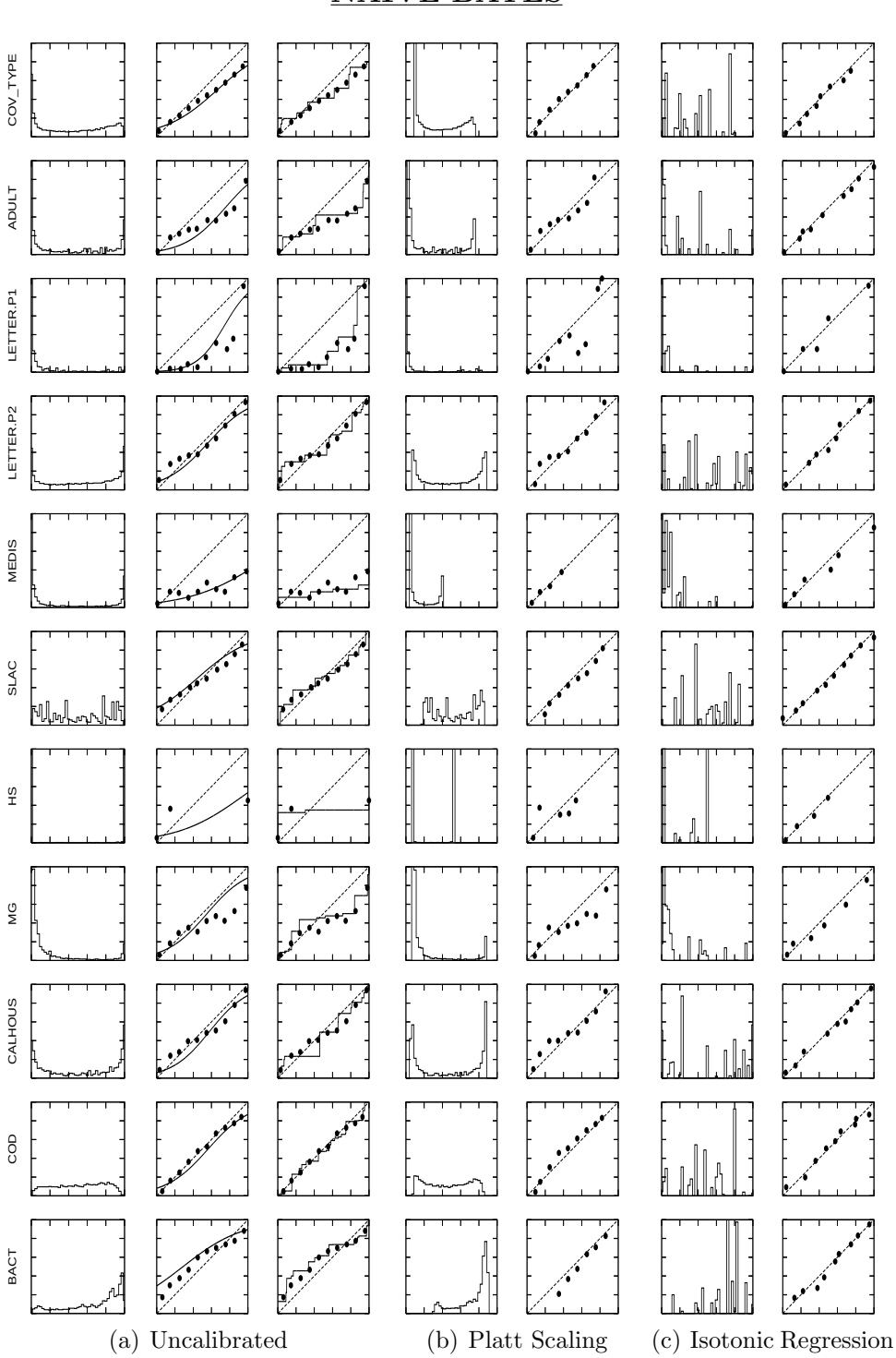


Figure 3.22: Histograms of predicted values and reliability diagrams for random forests before and after calibration.



(a) Uncalibrated (b) Platt Scaling (c) Isotonic Regression

Figure 3.23: Histograms of predicted values and reliability diagrams for memory based learning before and after calibration.



(a) Uncalibrated (b) Platt Scaling (c) Isotonic Regression

Figure 3.24: Histograms of predicted values and reliability diagrams for naive Bayes before and after calibration.

CHAPTER 4

ENSEMBLE SELECTION

4.1 Introduction

An ensemble is a collection of models whose predictions are combined by weighted averaging or voting. Dietterich (2000) states that "A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are accurate and diverse."

Many methods have been proposed to generate accurate, yet diverse, sets of models. Bagging (Breiman, 1996) trains models of one type (e.g., C4 decision trees) on bootstrap samples of the training set. Opitz (1999) bags features instead of training points. Boosting (Schapire, 2001) generates a potentially more diverse set of models than bagging by weighting the training set to force new models attend to those points that are difficult to classify correctly. Sullivan et al. (2000) boost features instead of training points. Error-correcting-output-codes (ECOC) (Dietterich & Bakiri, 1995) creates models with decorrelated errors by training models for multi-class problems on different dichotomies. Munro and Parmanto (1996) created diverse neural nets via competition among nodes.

Here we generate diverse sets of models by using many different algorithms. We use Support Vector Machines (SVMs), artificial neural nets (ANNs), memory-based learning (KNN), decision trees (DT), bagged decision trees (BAG-DT), boosted decision trees (BST-DT), boosted decision stumps (BST-STMP), random forests (RF), naive bayes (NB) and logistic regression (LOGREG). For each algorithm we train models using many different parameter settings. For example, we train 121 SVMs by varying the margin parameter C, the kernel, and the kernel parameters (e.g. varying gamma with RBF kernels.)

We train about 2000 models for each problem. Some models have excellent performance, equal to or better than the best models reported in the literature. Other models, however, have mediocre or even poor performance. Rather than combine good and bad models in an ensemble, we use forward stepwise selection from the library of models to find a subset of models that, when averaged together, yield excellent performance. The basic ensemble selection procedure is very simple:

1. Start with the empty ensemble.
2. Add to the ensemble the model in the library that maximizes the ensemble's performance on the error metric on a hillclimb (validation) set.
3. Repeat Step 2 for a fixed number of iterations or until all the models have been used.
4. Return the ensemble from the nested set of ensembles that has maximum performance on the hillclimb (validation) set.

Models are added to an ensemble by averaging their predictions with the models already in the ensemble. This makes adding a model to the ensemble *very* fast, allowing ensembles with excellent performance to be found in minutes from libraries with 2000 models. Moreover, the selection procedure allows us to optimize the ensemble to any easily computed performance metric. We evaluate the performance of ensemble selection on eight performance metrics. We believe this is the first time a learning method has been evaluated across such a wide variety of performance metrics.

On each performance metric we compare ensemble selection to the model in the library that performs best on that metric. Because we generate so many different models, libraries usually contain a few models with excellent performance on any performance metric. Just selecting the best single model from a library yields remarkably good performance. Ensemble selection, however, finds ensembles that

outperform the best single models. This suggests that using different learning methods and parameter settings generates libraries containing a diverse set of good-performing models.

4.2 Improving Ensemble Selection

The simple forward model selection procedure presented in the Introduction is fast and effective, but sometimes overfits to the hillclimbing set, reducing ensemble performance. We made three additions to this selection procedure to reduce overfitting. These are discussed in the next three sub-sections. These methods may be useful in other applications where forward stepwise selection is prone to overfitting, such as in feature selection (Kohavi & John, 1997).

4.2.1 Selection with Replacement

With model selection *without* replacement, performance improves as the best models are added to the ensemble, peaks, and then quickly declines. Performance drops because the best models in the library have been used and selection must now add models that hurt the ensemble. Figure 4.1 shows this behavior for root-mean-squared-error. Unfortunately, most error metrics yield much bumpier graphs than this when hillclimbing is done with small data sets, making it difficult to reliably pick a good stopping point. The loss in performance can be significant if the peak is missed.

Figure 4.1 also shows that selecting models *with replacement* greatly reduces this problem. Selection with replacement allows models to be added to the ensemble multiple times. Once peak performance is reached, if the unused models all hurt ensemble performance, selection adds models that were added before rather

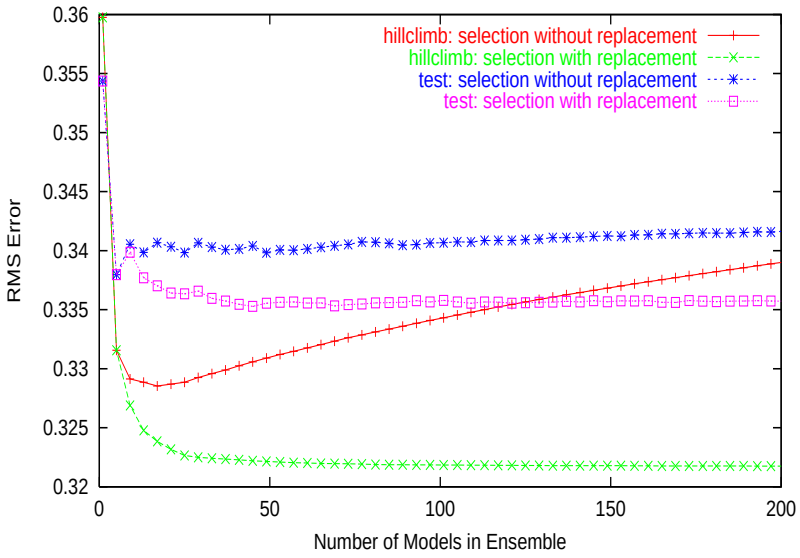


Figure 4.1: Selection With and Without Replacement.

than hurt performance. This flattens the performance curve past the peak, and allows selection to fine tune ensembles by weighting models: models added to the ensemble multiple times receive more weight.

Selection with replacement flattens the curve so much that a test set is not needed to determine when to stop adding models to the ensemble. The hillclimbing set can be used to stop hillclimbing. This means ensemble selection does not need more test sets than the base-level models would have used to select model parameters. Ensemble selection uses the validation set to do both parameter *and* model selection.

4.2.2 Sorted Ensemble Initialization

Forward selection sometimes overfits early in selection when ensembles are small. One way to prevent this is to initialize ensembles with more models. Instead of

starting with an empty ensemble, sort the models in the library by their performance, and put the best N models in the ensemble. N is chosen by looking at performance on the hillclimbing set. This typically adds 5-25 of the best models to an ensemble *before* greedy stepwise selection begins. Since each of the N best models performs well, they form a strong initial ensemble and it is more difficult for greedy selection to find models that overfit when added to the ensemble.

4.2.3 Bagged Ensemble Selection

As the number of models in a library increases, the chances of finding combinations of models that overfit the hillclimbing set increases. Bagging can minimize this problem. We reduce the number of models selection can choose from by drawing a random sample of models from the library and selecting from that sample. If a particular combination of M models overfits, the probability of those M models being in a random bag of models is less than $(1-p)^M$ for p the fraction of models in the bag. We use $p = 0.5$, and bag ensemble selection 20 times to insure that the best models will have many opportunities to be selected. The final ensemble is the average of the 20 ensembles. Bags of ensembles seem complex, but each ensemble is just a weighted average of models, so the average of a set of ensembles also is a simple weighted average of the base-level models.

4.3 Experimental Evaluation

4.3.1 Methodology

Models trained by different learning algorithms do not necessarily "speak the same language". A prediction of 0.14 from a neural net does not necessarily mean the

same thing as a prediction of 0.14 from a boosted tree or SVM. Predictions from neural nets often are well-calibrated posterior probabilities, but predictions from SVMs are just normalized distances to the decision surface. Averaging predictions from models that are not on commensurate scales may hurt ensemble performance. Because of this, we also calibrate the classifiers in the model library via the Platt Scaling method described in Section 3.2.1. Unless otherwise specified, we the model libraries include two versions of each classifier: one where the classifier has been calibrated, and one where the classifier is not calibrated. Here the ensemble selection hillclimb set is used for calibration as well.

Note that we do not determine what parameters yield best performance when training models. All models are added to a library no matter how good or bad they are. Model predictions on the train and hillclimbing sets are cached. This simplifies working with the library and makes model selection faster because the models do not have to be executed during selection.

The learning methods and parameter values we use to generate the model libraries are described in Appendix 4.B. The eleven data sets that we use for the evaluation are described in Appendix 4.A.

Performance Metrics

The eight performance metrics we use can be divided into three groups: threshold metrics, ordering/rank metrics and probability metrics.

The threshold metrics are accuracy (ACC), F-score (FSC) and lift (LFT). For thresholded metrics, it is not important how close a prediction is to a threshold, only if it is above or below threshold. See Giudici (Giudici, 2003) for a description of Lift Curves. Usually ACC and FSC have a fixed threshold (we use 0.5). For lift, often a fixed percent, p, of cases are predicted as positive and the rest as negative (we use $p = 25\%$).

The ordering/rank metrics depend only on the ordering of the cases, not the actual predicted values. As long as ordering is preserved, it makes no difference if predicted values fall between 0 and 1 or 0.89 and 0.90. These metrics measure how well the positive cases are ordered before negative cases and can be viewed as a summary of model performance across all possible thresholds. The rank metrics we use are area under the ROC curve (ROC), average precision (APR), and precision/recall break even point (BEP). See Provost and Fawcett (Provost & Fawcett, 1997) for a discussion of ROC from a machine learning perspective.

The probability metrics are minimized (in expectation) when the predicted value for each case coincides with the true conditional probability of that case being positive class. The probability metrics are squared error (RMS) and cross-entropy (MXE).

A description of all the performance metrics is available in Appendix 4.C

Comparing Across Performance Metrics

To permit averaging across metrics and problems, performances must be placed on comparable scales. We scale performance for each problem and metric from 0 to 1, where 0 is baseline performance and 1 is the best performance achieved by any model or ensemble. We use the following baseline model: predict p for every case, where p is the percent of positives in the data.

One disadvantage of normalized scores is that recovering a raw performance requires knowing what performances define the top and bottom of the scale, and as new best models are found the top of the scale may change. The numbers defining the normalized scales can be found in Appendix 4.D.

Table 4.1: Performance with and without model calibration. The best score in each column is bolded.

	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN
ES-BOTH	**.920**	.888	.967	**.982**	**.972**	.964	.932	**.944**	**.946**
ES-NOCAL	.919	**.897**	.967	.982	.970	.965	.912	.925	.942
ES-CAL	.912	.847	**.969**	.981	.969	**.966**	**.935**	.940	.940
BAYESAVG-BOTH	.893	.814	.964	.978	.963	.956	.918	.934	.928
BAYESAVG-CAL	.889	.820	.962	.977	.960	.955	.912	.925	.925
MODSEL-BOTH	.871	.861	.939	.973	.948	.938	.901	.916	.918
MODSEL-CAL	.870	.819	.943	.973	.948	.940	.892	.910	.912
MODSEL-NOCAL	.871	.858	.939	.973	.948	.938	.861	.871	.907
BAYESAVG-NOCAL	.875	.784	.955	.968	.953	.941	.874	.892	.905

4.3.2 Empirical Results

In this section we evaluate the performance of ensemble selection and we compare it with the performance obtained by selecting the best model from the library. We also compare against the performance of Bayesian model averaging (Domingos, 2000), a widely used ensemble learning method.

Table 4.1 shows the performance of ensemble selection (ES), model selection (MODSEL),[1] and Bayesian model averaging (BAYESAVG), with and without calibrated models. Results are shown for three different model libraries: 1) only uncalibrated models (NOCAL), 2) only calibrated models (CAL), and 3) both calibrated and uncalibrated models (BOTH). Each entry is the average of five folds on each of the eleven problems. The last column shows the mean performance over all eight metrics. Rows are sorted by mean performance. For the results in this subsection we used a hillclimb set of 1000 points.

Comparing results for ensemble selection with and without calibration (ES-CAL and ES-NOCAL), we see that calibrating models improves RMS and MXE (significant at .05) but hurts FSC. There is little difference for LFT, ROC, APR and BEP. For model selection we see the same trends: calibrated models yield

[1]Model selection chooses the best single model using the hillclimb set.

better RMS and MXE and worse FSC. The magnitudes of the differences suggest that most if not all of the improvement in RMS and MXE for ensemble selection with calibrated models is due to having better models in the library rather than from ensemble selection taking advantage of the common scale of the calibrated models. We are not sure why calibration makes FSC performance worse for both MODSEL and ES, but again suspect that the differences between ES-CAL and ES-NOCAL are due to differences in the performance of the base-level models.

Having both calibrated and uncalibrated models in the library (ES-BOTH and MODSEL-BOTH) gives the best of both worlds: it alleviates the problem with FSC while retaining the RMS and MXE improvements.

Unlike with ensemble selection, using calibrated models for Bayesian model averaging improves performance on all metrics, not just RMS and MXE (significant at .05). With calibrated models, Bayesian averaging outperforms model selection but is still not as good as ensemble selection. Having both calibrated and uncalibrated models (BAYESAVG-BOTH) also provides a small improvement for Bayesian model averaging.

For the rest of the chapter, we will use libraries that contain both calibrated and uncalibrated models, since they give the best performance for both ensemble selection and model selection.

4.3.3 Analysis of Training Size

Since the data used for hillclimbing is data taken away from training the individual models, keeping the hillclimb set small is important. Smaller hillclimb sets, however, are easier to overfit to, particularly when there are many models from which to select.

To explore ensemble selection's sensitivity to the size of the hillclimb set, we

ran ensemble selection with hillclimb sets containing 100, 250, 500, 1000, 2500, 5000, and 10000 data points. In each run we randomly selected the points for the hillclimb set and used the remainder for the test set. The hyperspectral and medis data sets contained too few points to leave sufficient test sets when using a 10K hillclimbing set and were omitted. Due to time constraints and the cost of generating the learning curves, we only used one random sample at each size and did not repeat the experiment.

Figure 4.2 shows learning curves for our eight performance measures and their mean. Each graph is an average over 9 problems. The x-axis uses a logscale to better show what happens with small hillclimbing sets. Normalized performance scores are plotted on the y-axis. For comparison, the graphs include the performance achieved by picking the single best model (MODSEL).

Unsurprisingly, the performance achieved with both ensemble selection and model selection using only 100 points for hillclimbing is quite bad. As data increases, both methods do better as they overfit less. Interestingly, ensemble selection is hurt less by a small hillclimbing set than model selection, suggesting that it is less prone to overfitting than model selection. Because of this, the benefit of ensemble selection over the best models appears to be strongest when training data is scarce. As the size of the hillclimbing sets goes from 1k to 10k, ensemble selection maintains its edge over model selection.

With small hillclimb sets, using bagging with ensemble selection is crucial to getting good performance; without it, mean performance using a 100 point hillclimb set drops from 0.888 to 0.817. In contrast, bagging provides very little if any benefit when a very large hillclimb set is used (more than 5000 points with our data sets).

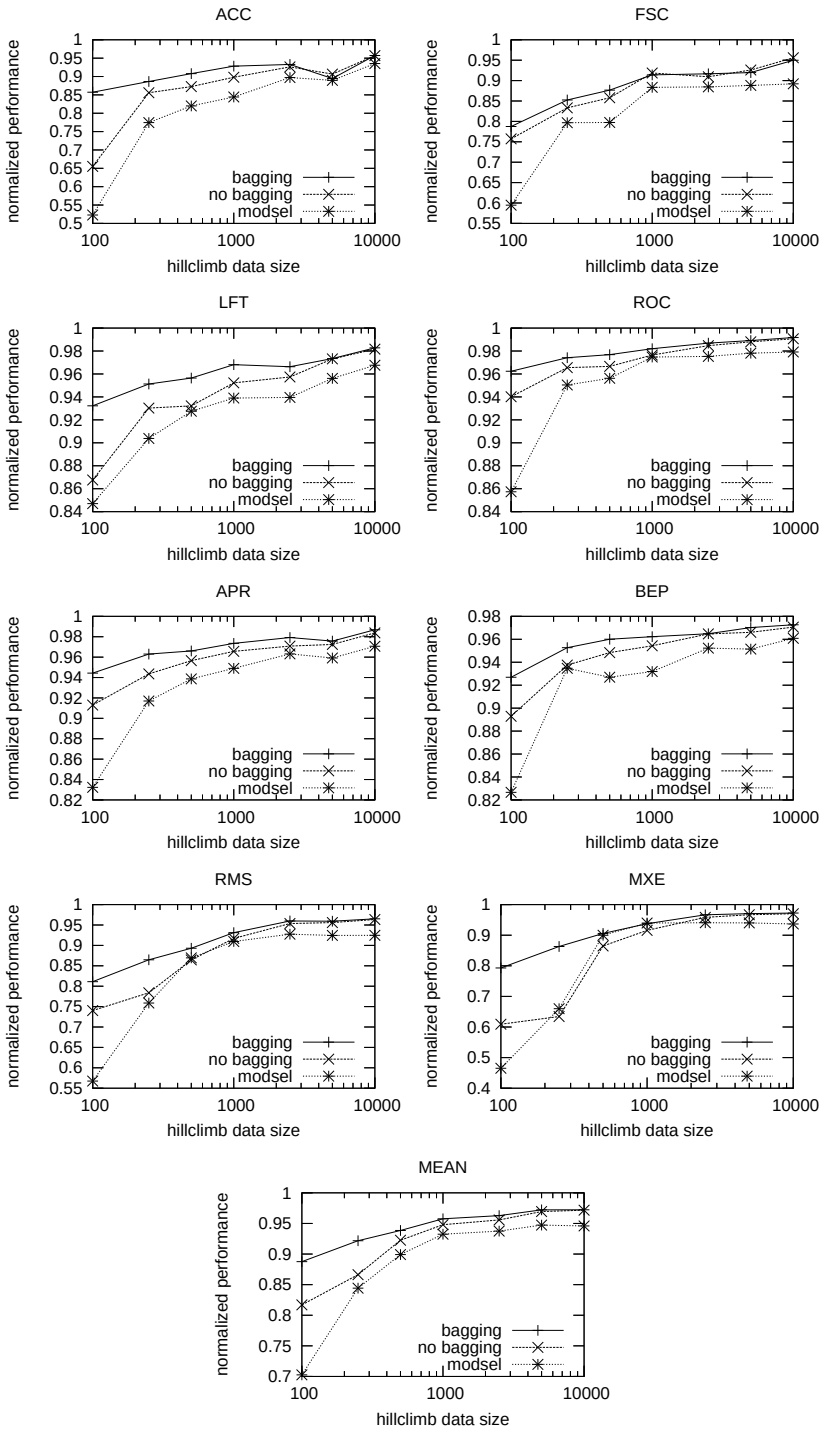


Figure 4.2: Learning curves for ensemble selection with and without bagging, and for picking the best single model (modsel).

4.3.4 Cross-Validated Ensemble Selection

It is clear from the results in Section 4.3.3 that the larger the hillclimb set, the better ensemble selection's performance will be. To maximize the amount of available data, we apply cross-validation to ensemble selection. Simply wrapping cross-validation around ensemble selection, however, will not help because the algorithm will still have just a fraction of the training data available for hillclimbing. Instead, we embed cross-validation within ensemble selection so that all of the training data can be used for the critical ensemble hillclimbing step. Conceptually, the procedure makes *cross-validated models*, then runs ensemble selection the usual way on a library of cross-validated base-level models.

A cross-validated model is created by training a model for each fold *with the same model parameters*. If there are 5 folds, there will be 5 individual models (each trained on 4000 points) that are 'siblings'; these siblings should only differ based on variance due to their different training samples. To make a prediction for a *test* point, a cross-validated model simply averages the predictions made by each of the sibling models. The prediction for a *training* point (that subsequently will be used for ensemble hillclimbing), however, only comes from the individual model that did not see the point during training. In essence, the cross-validated model delegates the prediction responsibility for a point that will be used for hillclimbing to the one sibling model that is not biased for that point.

Selecting a cross-validated model, whether during model selection or ensemble selection, means choosing *all* of the sibling models as a unit. If 5-fold cross-validation is used, selection chooses groups containing 5 sibling models at a time. In this case, when selection adds a cross-validated model to a growing ensemble, it really adds 5 different models of the same model type to the ensemble, each of which receives the same weight in the ensemble average.

Table 4.2: Performance with and without cross-validation for ensemble selection and model selection.

	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN
ES-CV	**.935**	**.926**	**.982**	**.996**	**.992**	**.977**	**.984**	**.989**	**.973**
MODSEL-CV	.907	.923	.971	.985	.968	.963	.945	.961	.953
ES	.920	.888	.967	.982	.972	.964	.932	.944	.946
MODSEL	.871	.861	.939	.973	.948	.938	.901	.916	.918

Table 4.3: Percent loss reduction by dataset.

	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN
ADULT	2.77	5.89	8.72	7.45	6.70	7.58	2.26	4.08	5.68
BACT	2.08	3.83	16.42	4.13	5.49	1.76	1.42	4.15	4.91
CALHOUS	7.95	9.49	48.00	8.69	8.81	6.15	7.17	12.74	13.63
COD	5.73	7.46	14.33	9.14	10.52	7.11	2.39	3.79	7.56
COVTYPE	6.68	7.26	12.35	11.34	14.99	7.64	7.80	12.92	10.12
HS	13.66	16.36	12.32	37.53	37.78	16.77	12.65	27.43	21.81
LETTER.P1	21.55	25.66	0.29	69.10	45.29	19.25	19.59	34.58	29.41
LETTER.P2	15.21	14.50	100	32.84	33.05	15.85	17.13	29.47	32.26
MEDIS	2.77	-0.05	2.08	6.33	7.28	4.62	1.40	2.70	3.39
MG	4.45	1.98	4.25	11.84	12.65	6.04	2.57	6.10	6.23
SLAC	2.49	3.27	13.65	6.92	9.62	2.73	1.66	3.33	5.46
MEAN	7.76	8.70	21.13	18.67	17.47	8.68	6.91	12.84	12.77
MEAN^{cv}	2.89	3.07	10.82	9.97	9.37	2.84	2.54	4.22	5.71

We ran ensemble selection with 5-fold cross-validation; this is analogous to normal ensemble selection with a 5000 point hillclimb set. Table 4.2 shows the results averaged over all the problems. Not only does cross-validation greatly improve ensemble selection performance, it also provides the same benefit to model selection. Five-fold cross-validated model selection actually outperforms non-cross-validated ensemble selection by a small but noticeable amount. However, ensemble selection with embedded cross-validation continues to outperform model selection.

Table 4.3 provides a different way to look at the results. The numbers in the table (except for the last row) are the percent reduction in loss of cross-validated ensemble selection, relative to non-cross-validated model selection. For example, if model selection achieves a raw accuracy score of 90%, and cross-validated ensemble

selection achieves 95% accuracy, then the percent reduction in loss is 50% —the loss has been reduced by half. The MEAN row is the average improvement for each metric, across datasets.

Embedding cross-validation within ensemble selection doubles its benefit over simple model selection (from 6.90% to 12.77%). This is somewhat of an unfair comparison; if a cross-validated model library is available, it is just as easy to do cross-validated model selection as it is to do cross-validated ensemble selection. The last row in Table 4.3 shows the percent loss reduction of cross-validated ensemble selection compared to cross-validated model selection.

While training five times as many models is computationally expensive, it may be useful for domains where the best possible performance is needed. Potentially more interesting, in domains where labeled data is scarce, cross-validated ensemble selection is attractive because a) it does not require sacrificing part of the training data for hillclimbing, b) it maximizes the size of the hillclimbing set (which Figure 4.2 shows is critical when hillclimb data is small), and c) training the cross-validated models is much more feasible with smaller training data.

Embedding cross-validation within ensemble selection significantly increases the performance of ensemble selection. There are two factors that could explain this increase in performance. First, the bigger hillclimbing set could make selecting models to add to the ensemble more reliable and thus make overfitting harder. Second, averaging the predictions of the sibling models could provide a bagging-like effect that improves the performance of the base-level models. To tease apart the benefit due to each of these factors we perform two additional experiments.

In one experiment, we use the same hillclimbing set as cross-validated ensemble selection, but instead of averaging the predictions of the sibling models, we use only the predictions of *one* of the siblings. Using this procedure we construct

Table 4.4: Breakdown of improvement from cross-validation.

	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN
ES-HILL	32.9	37.2	48.0	38.8	40.8	19.4	55.1	56.7	41.1
ES-AVG	80.5	13.6	54.0	59.0	55.7	77.4	46.8	51.8	54.9
SUM	113.4	50.8	102.0	97.8	96.5	96.8	101.9	108.5	96.0

five ensemble models, one for each fold, and report their mean performance. This provides a measure of the benefit due to the increase in the size of the hillclimb set (from cross-validation) while eliminating the bagging-like effect due to sibling model averaging.

In the other experiment, we use the smaller hillclimb sets used by *un-cross-validated* ensemble selection, but we do average the predictions of the sibling models. We again construct five ensemble models, one for each fold, and report their mean performance. This allows us to identify the performance increase due to the bagging-like effect of averaging the predictions of the sibling models.

Table 4.4 shows the results of these experiments. Entries in the table show the improvement provided by using a larger hillclimb set (ES-HILL) and by averaging the sibling models (ES-AVG) as a percentage of the total benefit of cross-validated ensemble selection. For example, looking at the ACC column, increasing the size of the hillclimb set from 1k to 5k yields a benefit equal to 32.9% of the total benefit provided by cross-validated ensemble selection, and averaging the sibling models yields a benefit equal to 80.5%.

The third row in the table is the sum of the first two rows. If the sum is lower than 100% the effects from ES-HILL and ES-AVG are super-additive, i.e. combining the two effects provides more benefit than the sum of the individual improvements. If the sum is higher than 100% then the two effects are sub-additive. For ACC, the sum is 113.4%, indicating that the effects of these two factors are sub-additive: the total performance is slightly less than would be expected if the

factors were independent. Except for the high variance metrics, FSC and ACC, the sums are close to 100%, indicating that the two effects are nearly independent.

The learning curves in Figure 4.2 suggest that increasing the size of the hillclimb set from 1k to 5k would explain almost all of the benefit of cross-validation. These results, however, show that on average across the eight metrics the benefit from ES-HILL and ES-AVG are roughly equal. About half of the benefit from embedding cross-validation within ensemble selection appears to result from the increase in the size of the hillclimb set, and the other half appears to result from averaging the sibling models. Increasing the size of the hillclimb set *via cross-validation* (as opposed to having more data available for hillclimbing) provides less benefit in practice because there is a mismatch between the base-level models used to make predictions on the hillclimbing set and the sibling-averaged models that will be used in the ensemble. In other words ensemble selection is hillclimbing using slightly different models than the ones it actually adds to the ensemble.

4.3.5 Direct Metric Optimization

One interesting feature of ensemble selection is its ability to build an ensemble optimized to an arbitrary metric. To test how much benefit this capability actually provides, we compare ensemble selection that optimizes the target metric with ensemble selection that optimizes a predetermined metric *regardless of the target metric*. For each of the eight metrics, we train an ensemble that optimizes it and evaluate the performance on all metrics. Optimizing RMS or MXE yields the best results.

Table 4.5 lists the performance of ensemble selection for a) always optimizing to RMS, b) always optimizing to MXE, and c) optimizing the true target metric (OPTMETRIC). When cross-validation is not used, there is modest benefit to

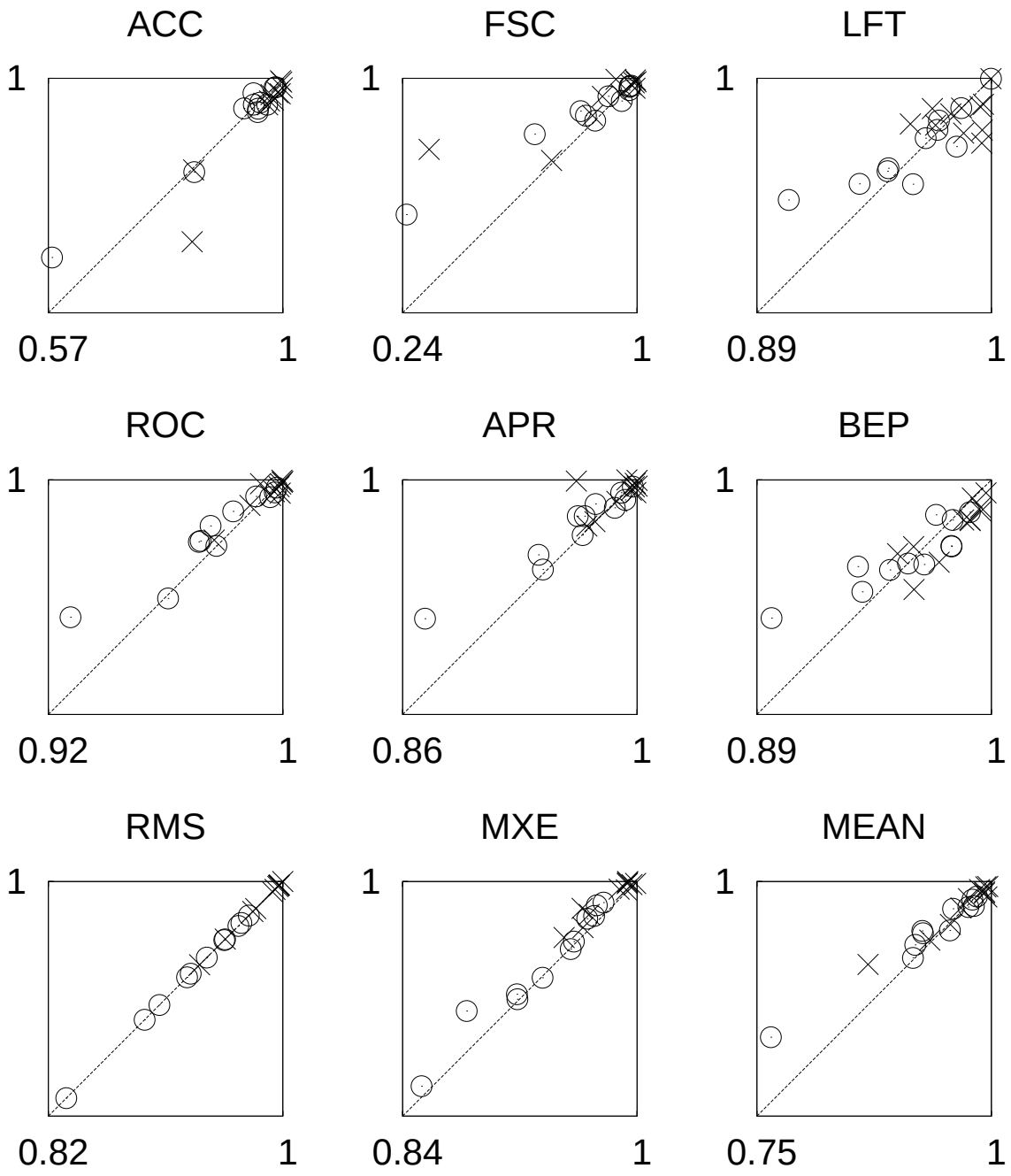


Figure 4.3: Scatter plots of ensemble selection performance when RMS is optimized (x-axis) vs when the target metric is optimized (y-axis). Points above the line indicate better performance by optimizing to the target metric (e.g. accuracy) then when optimizing RMS. Each point represents a different data set; circles are averages for a problem over 5 folds, and X's are performances using cross-validation. Each metric (and the mean across metrics) is plotted separately.

Table 4.5: Performance of ensemble selection when forced to optimize to one set metric.

	RMS	MXE	OPTMETRIC
ES-BOTH-CV	0.969	0.968	0.973
ES-BOTH	0.935	0.936	0.946

optimizing to the target metric. With cross-validation, however, the benefit from optimizing to the target metric is significantly smaller.

The scatter plots in Figure 4.3 plot the performance of optimizing to RMS against the performance of optimizing to the target metric, with one graph per target metric. Again, we can see that ensemble selection performs somewhat better when optimizing the target metric. Always optimizing RMS, however, is frequently very competitive, especially when performance gets close to a normalized score of 1. This is why the benefit of direct metric optimization is so small for cross-validated ensemble selection. These results suggest that optimizing RMS (or MXE) may be a good alternative if the target metric is too expensive to use for hillclimbing.

4.3.6 Model Library Pruning[2]

Including a large number of base level models, with a wide variety of parameter settings, in the model library helps ensure that at least some of the models will have good performance regardless of the metric optimized. At the same time, increasing the number of available models also increases the risk of overfitting the hillclimb set. Moreover, some of the models have such poor performance that they are unlikely to be useful for any metric one would want to optimize. Eliminating these models should not hurt performance, and might help.

In this section we investigate ensemble selection's performance when employing

[2]This section presents work done by Art Munson

varying levels of library pruning. The pruning works as follows: the models are sorted by their performance on the target metric (with respect to the hillclimb set), and only the top $X\%$ of the models are used for ensemble selection. Note that this pruning is different from work on ensemble pruning (Margineantu & Dietterich, 1997; Street & Kim, 2001; Tsoumakas et al., 2005; Zhang et al., 2006; Martínez-Munoz & Suárez, 2006). This is a *pre-processing* method, while ensemble pruning *post-processes* an existing ensemble.

Figure 4.4 shows the effect of pruning for each performance metric, averaged across the 11 data sets and 5 folds using non-cross-validated ensemble selection with and without bagging. For comparison, flat lines illustrate the performance achieved by model selection (modsel) and non-pruned ensemble selection (es-both). The legend is shown in the ACC graph.

The figure clearly shows that pruning usually does not hurt ensemble selection performance, and often improves it. For ACC, LFT, and BEP pruned ensemble selection (the line with boxes) seems to yield the same performance as non-pruned ensemble selection. For the other metrics, pruning yields superior performance. Indeed, when using more than 50% of the models performance decreases. Interestingly, library pruning reduces the need for bagging, presumably by reducing the potential for overfitting. [3]

The graphs in Figure 4.4 show the *average* behavior across our 11 data sets. Ensemble selection's behavior under pruning may in fact vary when each data set is considered individually. Averaging across problems could hide different peak points. Figure 4.5 shows RMS performance for each of the problems.

Although performance starts to decline at different pruning levels for the differ-

[3]The *bagging* line at 100% does not always match the *es-both* line, even though these should be equivalent configurations. This is particularly evident for FSC, the highest variance metric. The sorting performed before pruning alters ensemble selection's model sampling, resulting in additional variance.

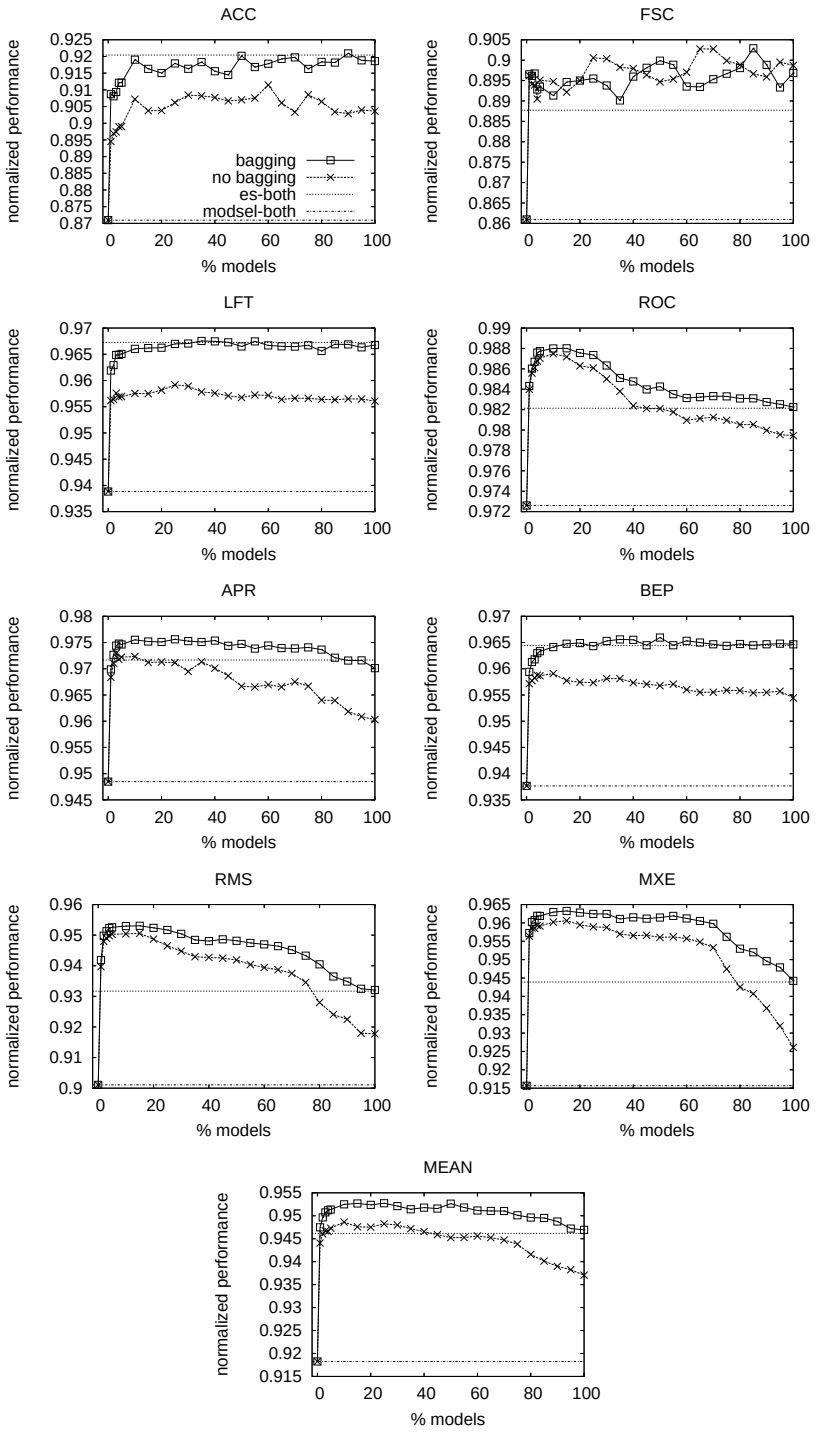


Figure 4.4: Pruned ensemble selection performance.

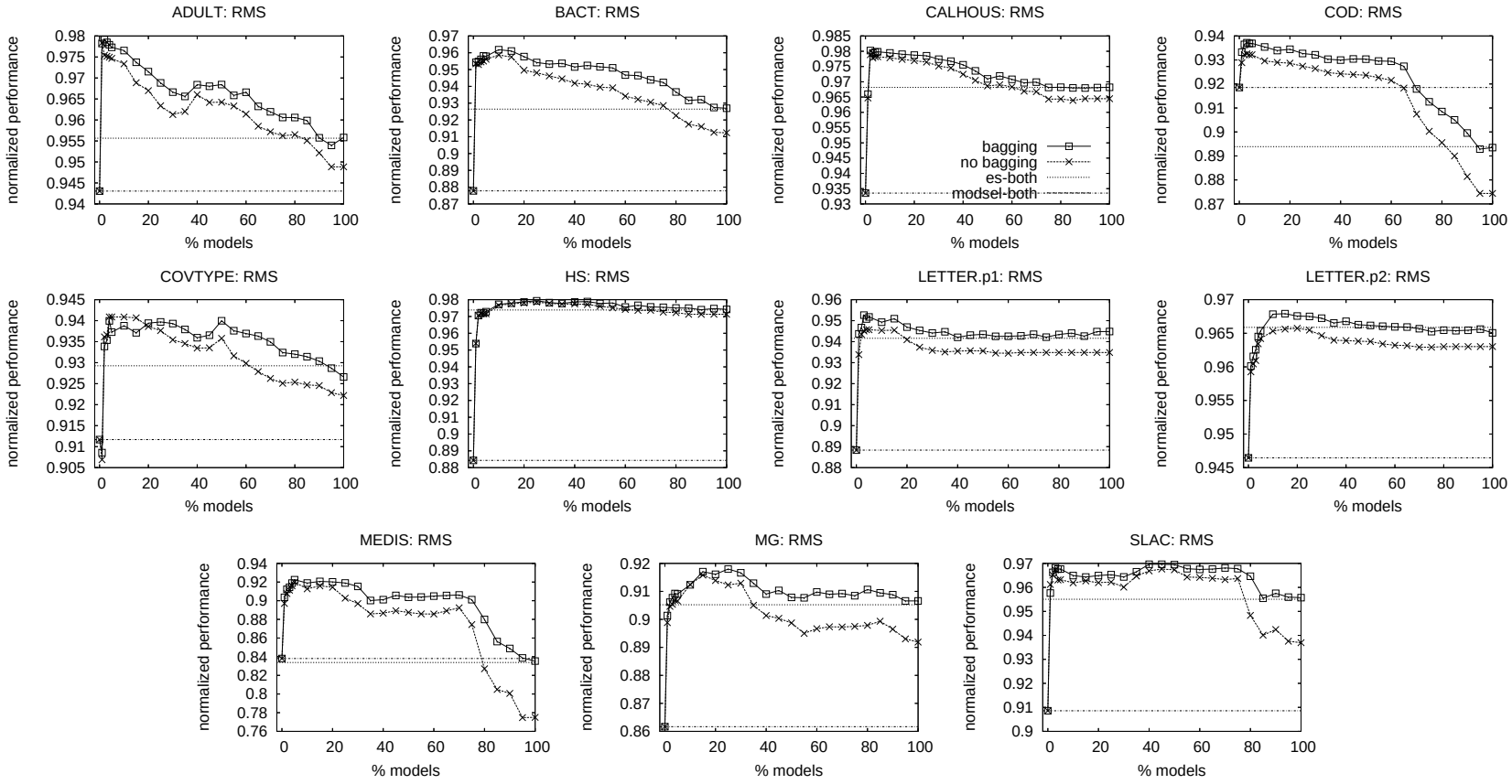


Figure 4.5: RMS performance for pruned ensemble selection.

ent problems, it is clear that larger model libraries increase the risk of overfitting the hillclimb set. Using 100% of the models is never worthwhile. At best, using the full library can match the performance of using only a small subset. In the worst case, ensemble selection overfits. This is particularly evident for the COD data set where model selection outperforms ensemble selection unless pruning is employed.

While further work is needed to develop good heuristics for automatically choosing an appropriate pruning level for a data set, simply using the top 10–20% models seems to be a good rule of thumb. An open problem is finding a better pruning method. For example, taking into account model diversity (see for example (Kuncheva & Whitaker, 2003; Peterson & Martinez, 2005)) might find better pruned sets.

4.4 Model Compression

While very accurate, the ensembles build by ensemble selection have one drawback: the space required to store all the base-level classifiers, and the time required to execute them at run-time, prohibits their use in applications where test sets are large (e.g. Google), where storage space is at a premium (e.g. PDAs), and where computational power is limited (e.g. hearing aids). In such situations, we propose using *model compression* (Bucila et al., 2006) to obtain fast and compact that perform almost as well as the ensembles build by ensemble selection.

The main idea behind model compression is to train a fast and compact model to approximate the function learned by a slow, large, but high performing model. Unlike the true function that is unknown, the function learned by the high performing model is available and can be used to label large amounts of synthetic *pseudo data*. A fast, compact and expressive model trained on enough synthetic

Table 4.6: Time in seconds to classify 10k cases.

	MIMIC	ENSEMBLE	ANN	SINGLE
ADULT	7.88	8560.61	3.94	48.31
COVTYPE	4.46	3440.99	1.05	37.31
HS	12.09	1817.17	3.85	3.85
LETTER.P1	2.59	1630.21	0.25	0.25
LETTER.P2	2.59	2651.95	0.74	526.34
MEDIS	4.78	190.18	2.85	2.85
MG	6.98	1220.04	1.80	53.58
SLAC	3.60	23659.03	2.85	74.48
AVERAGE	5.62	5396.27	2.17	93.37

data will not overfit and will closely approximate the function learned by the original model. This allows a slow, complex model such as a massive ensemble to be compressed into a fast, compact model with little loss in performance. For all the following results, the ensembles are compressed using neural networks with 256 hidden units trained on 400K pseudo data for each problem. The ensembles are trained to optimize RMS performance. Results are shown for only eight of the eleven datasets.

Table 4.6 shows the time in seconds required to classify 10,000 test cases for the mimic neural nets with 256 hidden units, the target ensemble models trained by ensemble selection, the best neural nets trained on the original 4k train set and the single best model in the ensemble library. There is significant variability in the speed of the best single model because different kinds of models are best for different problems and some of the models (e.g. boosted trees) are much more expensive than others (e.g. logistic regression). As expected, the ensemble is extremely expensive. On average, the ensemble takes about 0.5 seconds to classify a single training case (on a single workstation) and, on the SLAC problem, it takes 2.4 seconds per test case! The mimic neural nets, however, are very fast and take on average only about 0.5 milliseconds per test case.[4]

[4]The speed of different models depends significantly on how they are implemented. The times reported here are for typical implementations. For example, we use the SNNS neural net package

Table 4.7: Size of the models in MB.

	MIMIC	ENSEMBLE	ANN	SINGLE
ADULT	0.45	1234.72	0.22	3.95
COVTYPE	0.23	1108.16	0.03	3.41
HS	0.79	74.37	0.12	0.12
LETTER.P1	0.08	1.23	0.01	0.01
LETTER.P2	0.08	325.80	0.04	0.07
MEDIS	0.27	5.24	0.14	0.14
MG	0.50	25.75	0.03	3.25
SLAC	0.25	1627.08	0.13	0.30
AVERAGE	0.33	550.29	0.09	1.41

Table 4.8: RMSE results.

	MIMIC	ENSEMBLE	ANN	SINGLE	RATIO
ADULT	0.325	0.317	0.328	0.319	0.29
COVTYPE	0.340	0.334	0.378	0.349	0.84
HS	0.204	0.213	0.231	0.231	1.47
LETTER.P1	0.075	0.075	0.092	0.092	1.01
LETTER.P2	0.179	0.178	0.228	0.203	0.98
MEDIS	0.277	0.278	0.279	0.279	2.29
MG	0.288	0.287	0.295	0.290	0.88
SLAC	0.422	0.424	0.428	0.427	1.69
AVERAGE	0.264	0.263	0.282	0.274	0.97

Table 4.7 shows the size in megabytes for the different models in Table 4.8. A similar picture emerges as with execution times: on average ensembles are about 500 megabytes, about 500 times larger than the best single models, and the largest ensembles are more than a gigabyte. The mimic neural nets, however, are four times smaller than the best single models, and more than 1000 times smaller than the ensembles.

Table 4.8 shows, for eight of the problems, the raw RMS performance (not normalized scores) of the mimic neural nets, the target ensemble selection model, the best neural net trained on the original 4000 points training sets, and the best single model from the ensemble library. The performance of the mimic neural

(Zell et al., 1992), the IND decision tree package (Buntine & Caruana, 1991), the SVMlight SVM package (Joachims, 1999), etc. With care, some of these numbers probably could be improved by a factor of 10 or more, though we suspect the overall picture would not change substantially.

nets is as good as or better than the performance of the ensemble models they are trained to mimic on six of the eight problems, and always better than the performance of neural nets trained on the original 4k data.

The values in the last column of the table indicate how effective compression is at retaining the performance of the target ensemble selection models. These values are the ratio between the improvement in performance the mimic nets provide over the best neural nets and the improvement in performance the target ensemble selection models provide over the best neural nets. For example, if the mimic neural net has performance half way between the original neural net and the ensemble, the ratio is 0.5. If the mimic neural net has performance equal to the target ensemble, the ratio is 1.0. The only problem on which the ratio is less than 0.8 is ADULT. (The results on this problem are discussed in the next paragraph.) For a few problems the ratio is better than 1.0, indicating that the mimic neural net outperforms the ensemble. Note, however, that in two of the cases where the ratio is much larger than 1 (SLAC at 1.69 and MEDIS at 2.29), the range in performance is very small so this large ratio does not actually indicate a very large increase in performance. The ratio in the bottom row is the ratio calculated for the average RMSE performances in the table (not the average of the ratios, which would be inflated by the two problems with artificially high ratios). On average, model compression is able to achieve 97% of the performance increase that could at best be expected.

The only problem for which compression is ineffective is ADULT. On this problem the mimic net performs only a little better than a neural net trained on the original 4k data, and the mimic net does not perform as well as the best single model in the ensemble selection library. Interestingly, ADULT is the only data set that has high-arity nominal attributes. The three attributes with the highest arity

have 14, 16, 41 unique values. To train a neural net on ADULT, these attributes must first be converted to 14, 16, and 41 distinct binary attributes. The ADULT problem has only 14 attributes to begin with, yet these three attributes alone expand to 71 sparsely coded binary inputs. It is possible that neural nets are not well suited to this kind of problem, and this may prevent the mimic neural net from learning the ensemble target function. An alternate possibility is that the MUNGE procedure (Bucila et al., 2006) we used to generate pseudo data is not effective for this kind of problem.

For a more detailed treatment of model compression, additional results, and more discussion we direct the reader to (Bucila et al., 2006).

4.5 Conclusions

We presented ensemble selection, an ensemble learning method that uses forward stepwise selection from libraries of thousands of models to build ensembles that are optimized to the given performance metric. Using a variety of learning algorithms and parameter settings appears to be effective for generating libraries of diverse, high quality models. Experiments with eleven test problems and eight performance metrics show that ensemble selection consistently finds ensembles that outperform all other models, including models trained with bagging, boosting, and Bayesian model averaging.

Embedding cross-validation inside ensemble selection greatly increases its performance. Half of this benefit is due to having more data for hillclimbing; the other half is due to a bagging effect that results from the way cross-validation is embedded within ensemble selection. Unsurprisingly, reducing the amount of hillclimbing data hurts performance because ensemble selection can overfit this data more easily. In comparison to model selection, however, ensemble selection

seems much more resistant to overfitting when data is scarce. Further experiments varying the amount of *training data provided to the base-level models* are needed to see if ensemble selection is truly able to outperform model selection by such a significant amount on small data sets.

Counter to our and others' intuition (Duin, 2002), calibrating models to put all predictions on the same scale before averaging them did not improve ensemble selection's effectiveness. Most of calibration's improvement comes from the superior base-level models.

Our experiments show that directly optimizing to a target metric is better than always optimizing to some predetermined metric. That said, always optimizing to RMS or MXE was surprisingly competitive. These metrics may be good optimization proxies if the target metric is too expensive to compute repeatedly during hillclimbing.

Pruning the number of available models reduces the risk of overfitting during hillclimbing while also yielding faster ensemble building. In our experiments pruning rarely hurt performance and frequently improved it.

Finally, we showed that the main drawback of ensemble selection — that it builds ensembles that are very large and slow at test time — can be overcome with little to no loss by using model compression. Training small and fast neural net models to mimic the function learned by ensemble selection was able to retain more than 90% of the improvement provided by ensemble selection (over model selection), while being more than 1000 times smaller and 1000 times faster.

Acknowledgments

This is joint work with Rich Caruana. We thank Geoff Crew, Alex Ksikes, Charles Chiu and Kohsuke Kawaguchi for their help with the initial design of ensemble

selection, Art Munson for the help with the experimental evaluation, and Cristi Bucila for the help with model compression. We thank Tony Gualtieri for help with the HS data, C. Young et al. at Stanford Linear Accelerator for help with the SLAC data, and Foster Provost and Claudia Perlich for help with the BACT, COD and CALHOUS data. This work was supported by NSF Award 0412930.

BIBLIOGRAPHY

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

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

Bucila, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression: Making big, slow models practical. *Proc. of the 12th International Conf. on Knowledge Discovery and Data Mining (KDD'06)*.

Buntine, W., & Caruana, R. (1991). *Introduction to IND and recursive partitioning* (Technical Report FIA-91-28). NASA Ames Research Center.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *First International Workshop on Multiple Classifier Systems*, 1–15.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*.

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. *ICML* (pp. 223–230). Morgan Kaufmann, San Francisco, CA.

Duin, R. P. W. (2002). The combining classifier: To train or not to train? *ICPR (2)* (pp. 765–770).

Giudici, P. (2003). *Applied data mining*. New York: John Wiley and Sons.

Gualtieri, A., Chettri, S. R., Cromp, R., & Johnson, L. (1999). Support vector machine classifiers as applied to aviris data. *Proc. Eighth JPL Airborne Geoscience Workshop*.

Joachims, T. (1999). Making large-scale SVM learning practical. *Advances in Kernel Methods*.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*.

Kuncheva, L. I., & Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, *51*, 181–207.

Margineantu, D. D., & Dietterich, T. G. (1997). Pruning adaptive boosting. *ICML* (pp. 211–218). Morgan Kaufmann.

Martínez-Munoz, G., & Suárez, A. (2006). Pruning in ordered bagging ensembles. *ICML* (pp. 609–616). New York, NY, USA: ACM Press.

Munro, P., & Parmanto, B. (1996). Competition among networks improves committee performance. *Advances in Neural Information Processing Systems*.

Opitz, D. (1999). Feature selection for ensembles. *AAAI/IAAI* (pp. 379–384).

Perlich, C., Provost, F., & Simonoff, J. S. (2003). Tree induction vs. logistic regression: A learning-curve analysis. *J. Mach. Learn. Res.*, *4*, 211–255.

Peterson, A. H., & Martinez, T. R. (2005). Estimating the potential for combining learning models. *Proc. of the ICML Workshop on Meta-Learning* (pp. 68–75).

Provost, F., & Domingos, P. (2003). Tree induction for probability-based rankings. *Machine Learning*.

Provost, F. J., & Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. *Knowledge Discovery and Data Mining* (pp. 43–48).

Schapire, R. (2001). The boosting approach to machine learning: An overview. *In MSRI Workshop on Nonlinear Estimation and Classification*.

Street, W. N., & Kim, Y.-H. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. *KDD* (pp. 377–382).

Sullivan, J., Langford, J., Caruana, R., & Blum, A. (2000). Featureboost: A meta-learning algorithm that improves model robustness. *Proceedings of the Seventeenth International Conference on Machine Learning*.

Tsoumakas, G., Angelis, L., & Vlahavas, I. (2005). Selective fusion of heterogeneous classifiers. *Intelligent Data Analysis*, *9*, 511–525.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. San Francisco: Morgan Kaufmann. Second edition.

Zell, A., Mache, N., Huebner, R., Schmalzl, M., Sommer, T., & Korb, T. (1992). *SNNS: Stuttgart neural network simulator* (Technical Report). University of Stuttgart, Stuttgart.

Zhang, Y., Burer, S., & Street, W. N. (2006). Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, *7*, 1315–1338.

APPENDIX

4.A Data Sets

We use eleven binary classification data sets for our empirical evaluations. ADULT, COV_TYPE and LETTER are from the UCI Repository (Blake & Merz, 1998). COV_TYPE has been converted to a binary problem by treating the largest class as the positive and the rest as negative. We converted the 26-class LETTER data to boolean classification in two ways. LETTER.p1 treats "O" as positive and the remaining 25 letters as negative, yielding a very unbalanced problem. LETTER.p2 uses letters A-M as positives and the rest as negatives, yielding a well balanced, but more challenging, learning problem. HS is the IndianPine92 data set (Gualtieri et al., 1999) where the difficult class Soybean-mintill is the positive class. SLAC is a particle physics problem from the Stanford Linear Accelerator. MEDIS and MG are medical data sets. COD, BACT, and CALHOUS are three of the datasets used in (Perlich et al., 2003).

Three of these datasets, ADULT, COD, and BACT, contain nominal attributes. For ANNs, SVMs, KNNs, and LOGREG we transform the nominal attributes to boolean (one boolean per value). Each DT, BAG-DT, BST-DT, BST-STMP, RF, and NB model is trained twice, once with transformed attributes and once with the original ones. See Table 4.9 for characteristics of these problems.

4.B Learning Algorithms

We attempt to explore the space of parameters and common variations for each learning algorithm as thoroughly as is computationally feasible. This section summarizes the parameters used for each learning algorithm, and may safely be skipped by readers who are easily bored.

Table 4.9: Description of problems

PROBLEM	#ATTRIBUTES	TRAIN SIZE	TEST SIZE	%POZITIVES
ADULT	14/104	5000	35222	25%
BACT	11/170	5000	34262	69%
COD	15/60	5000	14000	50%
CALHOUS	9	5000	14640	52%
COV_TYPE	54	5000	25000	36%
HS	200	5000	4366	24%
LETTER.P1	16	5000	14000	3%
LETTER.P2	16	5000	14000	53%
MEDIS	63	5000	8199	11%
MG	124	5000	12807	17%
SLAC	59	5000	25000	50%

SVMs: we use the following kernels in SVMLight (Joachims, 1999): linear, polynomial degree 2 & 3, radial with width {0.001,0.005,0.01,0.05,0.1,0.5,1,2}. We also vary the regularization parameter by factors of ten from 10^{-7} to 10^{3} with each kernel.

ANN we train neural nets with gradient descent backprop and vary the number of hidden units {1,2,4,8,32,128} and the momentum {0,0.2,0.5,0.9}. We halt training the nets at many different epochs and use validation sets to select the best nets.

Logistic Regression (LOGREG): we train both unregularized and regularized models, varying the ridge (regularization) parameter by factors of 10 from 10^{-8} to 10^{4}.

Naive Bayes (NB): we use Weka (Witten & Frank, 2005) and try all three of the Weka options for handling continuous attributes: modeling them as a single normal, modeling them with kernel estimation, or discretizing them using supervised discretization.

KNN: we use 26 values of K ranging from $K = 1$ to $K = |trainset|$. We use KNN with Euclidean distance and Euclidean distance weighted by gain ratio. We also use distance weighted KNN, and locally weighted averaging. The kernel widths for locally weighted averaging vary from 2^{0} to 2^{10} times the minimum distance

between any two points in the train set.

Random Forests (RF): we tried both the Breiman-Cutler and Weka implementations; Breiman-Cutler yielded better results so we report those here. The forests have 1024 trees. The size of the feature set considered at each split is 1,2,4,6,8,12,16 or 20.

Decision trees (DT): we use different splitting criteria, pruning options, and smoothing (Laplacian or Bayesian smoothing). We use all of the tree models in Buntine's IND package (Buntine & Caruana, 1991): BAYES, ID3, CART, CART0, C4, MML, and SMML. We also generate trees of type C44LS (C4 with no pruning and Laplacian smoothing), C44BS (C44 with Bayesian smoothing), and MMLLS (MML with Laplacian smoothing). See (Provost & Domingos, 2003) for a description of C44LS.

Bagged trees (BAG-DT): we bag 100 trees of each type described above. With **boosted trees (BST-DT)** we boost each tree type as well. Boosting can overfit, so we consider boosted trees after 2,4,8,16,32,64,128,256,512,1024 and 2048 steps of boosting. With **boosted stumps (BST-STMP)** we boost single level decision trees generated with 5 different splitting criteria, each boosted for 2,4,8,16,32,64,128,256,512,1024,2048,4096,8192 steps.

With LOGREG, ANN, SVM and KNN we scale attributes to 0 mean 1 std. With DT, RF, NB, BAG-DT, BST-DT and BST-STMP we don't scale the data. In total, we train about 2000 different models in each trial on each problem.

4.C Performance Metrics Used

accuracy (ACC): probably the most widely used performance metric in Machine Learning. It is defined as the proportion of correct predictions the classifier makes relative to the size of the dataset. If a classifier has continuous outputs

(e.g. neural nets), a threshold is set and everything above this threshold is predicted to be a positive.

root-mean-squared-error (RMSE): widely used in regression, it measures how much predictions deviate from the true targets. [5]RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum(Pred(C) - True(C))^2} \tag{4.1}$$

mean cross entropy (MXE): is used in the probabilistic setting when interested in predicting the probability that an example is positive (1). It can be proven that in this setting minimizing the cross entropy gives the maximum likelihood hypothesis. mean cross entropy is defined as:

$$\begin{aligned} MXE = \quad & -\tfrac{1}{N}\sum(True(C) * ln(Pred(C)) + \\ & (1 - True(C)) * ln(1 - Pred(C))) \end{aligned} \tag{4.2}$$

(The assumptions are that $Pred(C) \in [0, 1]$ and $True(C) \in \{0, 1\}$)

receiver operating characteristic (ROC): has it's roots in WWII in the early days of radar where it was difficult to distinguish between true positives and false positives. ROC is a plot of sensitivity vs. (1-specificity) for all possible thresholds. Sensitivity is the defined as $P(Pred = positive|True = positive)$ and is approximated by the fraction of true positives that are predicted as positive (this is the same as recall). Specificity is $P(Pred = negative|True = negative)$. It is approximated by the fraction of true negatives predicted as negatives. AUC, the area under the ROC curve, is used as a summary statistic. ROC has a number of nice properties that make it more principled than similar measures such as average precision. AUC is widely used in fields such as medicine, and recently has become more popular in the Machine Learning community.

[5]Root-mean-squared error is applicable to binary classification settings where the classifier outputs predictions on $[0, 1]$ that are compared with the true target labels on $\{0, 1\}$.

lift (LFT): often used in marketing analysis, Lift measures how much better a classifier is at predicting positives than a baseline classifier that randomly predicts positives (at the same rate observed for positives in the data). The definition is:

$$LIFT = \frac{\%\text{of true positives above the threshold}}{\%\text{of dataset above the threshold}} \tag{4.3}$$

Usually the threshold is set so that a fixed percentage of the dataset is classified as positive. For example, suppose a marketing agent wants to send advertising to potential clients, but can only afford to send ads to 10% of the population. A classifier is trained to predict how likely a client is to respond to the advertisement, and the ads are sent to the 10% of the population predicted most likely to respond. A classifier with optimal lift will get as many clients as possible that will respond to the advertisement in this set.

precision and recall: These measures are widely used in Information Retrieval. Precision is the fraction of examples predicted as positive that are actually positive. Recall is the fraction of the true positives that are predicted as positives. These measures are trivially maximized by not predicting anything, or predicting everything, respectively, as positive. Because of this these measures often are used together. There are different ways to combine these measures as described by the next three metrics.

precision-recall F-score (FSC): for a given threshold, the F-score is the harmonic mean of the precision and recall at that threshold.

precision-recall break-even point (BEP): is defined as the precision at the point (threshold value) where precision and recall are equal.

average precision (APR): usually is computed as the average of the precisions at eleven evenly spaced recall levels.

4.D Performance Scales

Table 4.10 lists the performance numbers that determine the normalized scores. Each entry contains the baseline performance (bottom of the scale) and the best performance achieved by any model or ensemble (top of the scale).

Table 4.10: Scales used to compute normalized scores. Each entry shows bottom / top for the scale.

	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE
ADULT	0.752 / 0.859	0.398 / 0.705	1.000 / 2.842	0.500 / 0.915	0.248 / 0.808	0.248 / 0.708	0.432 / 0.312	0.808 / 0.442
BACT	0.692 / 0.780	0.818 / 0.855	1.000 / 1.345	0.500 / 0.794	0.692 / 0.891	0.692 / 0.824	0.462 / 0.398	0.891 / 0.697
CALHOUS	0.517 / 0.889	0.681 / 0.893	1.000 / 1.941	0.500 / 0.959	0.517 / 0.964	0.517 / 0.895	0.500 / 0.283	0.999 / 0.380
COD	0.501 / 0.784	0.666 / 0.796	1.000 / 1.808	0.500 / 0.866	0.499 / 0.864	0.499 / 0.782	0.500 / 0.387	1.000 / 0.663
COVTYPE	0.639 / 0.859	0.531 / 0.804	1.000 / 2.487	0.500 / 0.926	0.362 / 0.879	0.361 / 0.805	0.480 / 0.320	0.944 / 0.478
HS	0.759 / 0.949	0.389 / 0.894	1.000 / 3.656	0.500 / 0.985	0.243 / 0.962	0.241 / 0.898	0.428 / 0.198	0.797 / 0.195
LETTER.p1	0.965 / 0.994	0.067 / 0.917	1.000 / 4.001	0.500 / 0.999	0.036 / 0.975	0.035 / 0.917	0.184 / 0.067	0.219 / 0.025
LETTER.p2	0.533 / 0.968	0.696 / 0.970	1.000 / 1.887	0.500 / 0.996	0.534 / 0.997	0.533 / 0.970	0.499 / 0.157	0.997 / 0.125
MEDIS	0.893 / 0.905	0.193 / 0.447	1.000 / 2.917	0.500 / 0.853	0.108 / 0.462	0.107 / 0.469	0.309 / 0.272	0.491 / 0.365
MG	0.831 / 0.900	0.290 / 0.663	1.000 / 3.210	0.500 / 0.911	0.170 / 0.740	0.169 / 0.686	0.375 / 0.278	0.656 / 0.373
SLAC	0.501 / 0.726	0.667 / 0.751	1.000 / 1.727	0.500 / 0.813	0.501 / 0.816	0.501 / 0.727	0.500 / 0.420	1.000 / 0.755