PROTOTYPE SELECTION FOR COMPOSITE NEAREST NEIGHBOR CLASSIFIERS

A Dissertation Presented

by

DAVID B. SKALAK

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 1997

Department of Computer Science

PROTOTYPE SELECTION FOR COMPOSITE NEAREST NEIGHBOR CLASSIFIERS

A Dissertation Presented

by

DAVID B. SKALAK

Approved as to style and content by:

Edwina L. Rissland, Chair

Paul E. Utgoff, Member

Andrew G. Barto, Member

Michael Sutherland, Member

David W. Stemple, Department Chair
Computer Science

ACKNOWLEDGMENTS

Edwina Rissland has provided unflagging support during my decade in Amherst. More important than any single piece of research is the influence that a teacher has on one's research career and on one's life. I am fortunate to have worked under Edwina.

Andy Barto, Paul Utgoff and Mike Sutherland have each provided intellectual support of the highest rank.

I deeply thank Robert Constable, Chair of the Department of Computer Science at Cornell University, for providing me with computing facilities and office space.

For valuable assistance, I thank Paul Cohen, Paul Utgoff, Jamie Callan, the members of the Experimental Knowledge Systems Laboratory at UMass, Priscilla Coe, Sharon Mallory, Neil Berkman and Zijian Zheng.

I am especially grateful to Andy Golding.

Most of all, I thank Claire Cardie.

This work was supported in part by the Air Force Office of Scientific Research under Contract 90-0359, by National Science Foundation Grant No. EEC-9209623 State-University-Industry Cooperative Research on Intelligent Information Retrieval, by a University of Massachusetts Graduate School Fellowship, and by trading in S&P 100 Index options.

ABSTRACT

PROTOTYPE SELECTION FOR COMPOSITE NEAREST NEIGHBOR CLASSIFIERS

MAY 1997

DAVID B. SKALAK, B.S., UNION COLLEGE

M.A., DARTMOUTH COLLEGE

J.D., HARVARD LAW SCHOOL

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Edwina L. Rissland

Combining the predictions of a set of classifiers has been shown to be an effective way to create composite classifiers that are more accurate than any of the component classifiers. Increased accuracy has been shown in a variety of real-world applications, ranging from protein sequence identification to determining the fat content of ground meat.

Despite such individual successes, the answers are not known to fundamental questions about classifier combination, such as “Can classifiers from any given model class be combined to create a composite classifier with higher accuracy?” or “Is it possible to increase the accuracy of a given classifier by combining its predictions with those of only a small number of other classifiers?”. The goal of this dissertation is to provide answers to these and closely related questions with respect to a particular model class, the class of nearest neighbor classifiers.

We undertake the first study that investigates in depth the combination of nearest neighbor classifiers. Although previous research has questioned the utility of combining nearest neighbor classifiers, we introduce algorithms that combine a small number of component nearest neighbor classifiers, where each of the components stores a small number of prototypical instances. In a variety of domains, we show that these algorithms yield composite classifiers

that are more accurate than a nearest neighbor classifier that stores all training instances as prototypes.

The research presented in this dissertation also extends previous work on prototype selection for an independent nearest neighbor classifier. We show that in many domains, storing a very small number of prototypes can provide classification accuracy greater than or equal to that of a nearest neighbor classifier that stores all training instances. We extend previous work by demonstrating that algorithms that rely primarily on random sampling can effectively choose a small number of prototypes.

TABLE OF CONTENTS

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The Problem

Combining the predictions of a set of component classifiers has been shown to yield accuracy higher than the most accurate component on such disparate tasks as handwritten character recognition [Ho *et al.*, 1994], protein sequence identification [Zhang *et al.*, 1992], time series prediction [English, 1996], ground cover identification from satellite data [Benediktsson *et al.*, 1993] and the calculation of the fat content of ground meat [Thodberg, 1995]. Despite these varied individual successes, we do not know the answers even to the most fundamental questions about classifier combination, such as "Can classifiers from any given model class be combined to create a composite classifier with higher accuracy?" and "Is it possible to increase accuracy efficiently by combining the predictions of a very small number of classifiers?". The goal of this dissertation is to answer these and closely related questions with respect to a particular model class, the class of nearest neighbor classifiers.

Our lack of knowledge of how to combine classifiers is not due to lack of study. The combination of forecasts, particularly economic and weather predictions, has been studied for over a century [Clemen, 1989]. During the past several years, in a variety of application domains, researchers in machine learning, computational learning theory, pattern recognition and statistics have re-ignited the effort to learn how to create and combine an ensemble of classifiers. This research has the potential to apply accurate composite classifiers to real-world problems by intelligently combining known learning algorithms. One attraction of classifier combination is the possibility that by combining a set of "dumb" classifiers, we may be able to perform classification better than with the sophisticated algorithms now available. Another attraction is that the accuracy of a sophisticated classifier may be increased, merely by combining its predictions with those made by an unsophisticated classifier. These intriguing possibilities provide reasons to ask the questions that we will ask in this thesis.

Classifier combination falls within the *supervised learning* paradigm. This task orientation assumes that we have been given a set of training examples (also called training *instances*), which are customarily represented by feature vectors. Each training example is labeled with a class target, which is a member of a finite, and usually small, set of class labels. The goal of supervised learning is to predict the class labels of examples that have not been seen, and to do so accurately and efficiently. We use the term *predict* synonymously with *classify*, despite the apparent temporal component to a prediction.

A composite classifier first must be created and trained to perform supervised learning. For example, in one well-known composite classifier architecture, composite training begins by training each of the components. Then, a combining classifier learns to output the correct prediction based on the predictions of the components. When the trained composite classifier is presented with a new example of unknown class, each of the component classifiers is polled for its prediction of the example's class. The combining classifier amalgamates these component predictions to yield a prediction for the entire composite classifier. The combining algorithm may be as simple as voting, but may be a more sophisticated learning algorithm.

The intuition that combining classifiers might lead to improved classification accuracy can be supported by at least three powerful and general ideas: *expertise*, *consensus* and *diversification*. First, one component classifier may have expertise in some sub-region of the space of instances, displayed as high local accuracy. If a composite classifier can learn when each component is best in some region, it may display higher accuracy throughout the instance space by using the predictions of local experts.

Secondly, faith in group consensus may also support our intuition that combination may be effective: some decisions may be better made through group decision-making than by individual fiat. Whereas one individual may be prone to err, a group decision may be more reliable on average, or so this intuition suggests. Fortuitously, even the lack of complete consensus may also be telling. In general, where a unanimous decision has not been achieved, something may be learned from the fact that a specific individual, with a known history of decision-making success, has a different opinion on a given issue. This information might be exploited to make a correct decision. In a classification setting, an algorithm that combines the predictions of a set of components may learn from the fact that a particular classifier misclassifies some particular instances.

Third, diversification of computing resources also supports classifier combination. Rather than concentrating one's computational efforts on developing a single classifier — on which all classification bets will ride — perhaps it is better to hedge those bets. In this way, a portfolio of classifiers is analogous to a portfolio of diverse financial instruments. For financial portfolios, diversification is used to reduce the risk of catastrophic loss. To hedge intelligently, one might try to choose a set of classifiers that are "complementary" in some sense, and then combine their predictions. The effect of errors made by some classifiers may be mitigated by amalgamating their predictions. For instance, where real-valued predictions are made, these can be averaged, and applying an ensemble of classifier predictions may well reduce the variance of the ultimate prediction as well as improve the accuracy.

Nevertheless, our practical and theoretical understanding of classifier combination lags these intuitions, however well-supported. To make concrete progress, we propose the following hypothesis:

> *The accuracy of a standard nearest neighbor classifier can be improved or exceeded through composite classifier architectures that incorporate a number of component nearest neighbor classifiers.*

This statement is the general hypothesis examined in this dissertation. We will refine and restate the hypothesis in the next section, once we have discussed some of the design goals for classifier combination.

For the most part, we pursue our investigation within a straightforward approach to combining classifiers that has been termed *stacked generalization* [Wolpert, 1992; Wolpert, 1993]. In stacked generalization's most basic form, a layered architecture consists of a set of *component classifiers* that form the first layer, and of a single *combining classifier* that forms the second (somewhat degenerate) layer (Figure 1.1). Wolpert calls the component classifiers the *level-0 classifiers* and the combining classifier, the *level-1 classifier*. The composite classifier is called a *stacked generalizer*.

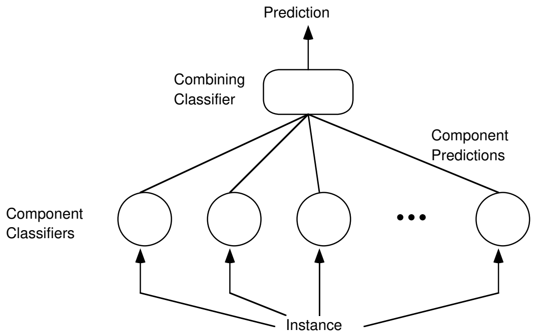

Figure 1.1: Composite classifier architecture

Stacked generalization is perhaps the most widely used and studied model for classifier combination. Nevertheless, all the basic open questions about composite classification remain open even in this restricted framework. Wolpert, who has done important recent work in this area, echoes the state of research:

> It is important to note that many aspects of stacked generalization are, at present, "black art." For example, there are currently no hard and fast rules saying what level[-]0 generalizers one should use, what level[-]1 generalizer one should use, what k numbers to use to form the level[-]1 input space, etc. [*sic*] In practice one must usually be content to rely on prior knowledge to make (hopefully) intelligent choices for how to set these details. [Wolpert, 1992, p.245].

Wolpert's observation is as true today as when it was made several years ago. In part, the problem is difficult because it involves one of the deepest problems of artificial intelligence: representation. In a stacked classifier, we argue that the role of the level-0 components is to provide the attributes for training and applying the level-1 learning algorithm. The level-1 learning algorithm learns to combine the predictions of the component classifiers. It learns to perform that task on the basis of a level-1 training set whose representation is determined by the component classifiers. Building an appropriate level-1 representation is especially difficult because we must design and rely on the output of the component classifiers to create the features, and the predictions made for out-of-sample data are not within our direct control.

To investigate classifier combination empirically, we must choose some model class or classes of component classifiers for a testbed. The first question is whether to combine classifiers from the same model class (*homogeneous* classifiers) or classifiers from different model classes (*heterogeneous* classifiers). While we postpone a discussion of this issue until the next chapter, for the purposes of this thesis, we choose homogeneous classifiers. Here we merely note that there have been a number of successful efforts at combining homogeneous classifiers, including multiple decision trees and multiple neural networks. Searching the more complex space of sets of heterogeneous component classifiers has not been shown necessary to achieve high accuracy.

We further focus this dissertation by restricting our investigation to combining instance-based classifiers, which are a subset of the class of *local classifiers* [Bottou and Vapnik, 1992; Atkeson *et al.*, 1996]. Atkeson, Moore and Schaal characterize local models as those "that only attempt to fit the training data in a region around the location of the query [i.e., the test instance]" [Atkeson *et al.*, 1996, p.2]. This class includes k-nearest neighbor algorithms and radial basis function networks, but does not include neural networks trained with backpropagation [Bottou and Vapnik, 1992], for example.

Instance-based classifiers, such as nearest neighbor classifiers, predict the class of an unseen instance by computing its similarity to a set of stored instances called *prototypes*. *Prototype selection* — storing a well-selected, proper subset of available training instances — has been shown to increase classifier accuracy in many domains. At the same time, using prototypes dramatically decreases storage and classification-time costs. Identifying prototypes may be a more expensive computation. Prototype selection will be useful only if the costs saved in applying an instance-based classifier outweigh the initial cost of identifying them. However, despite the demonstrated utility of prototype selection for individual instance-based classifiers, no algorithm has been developed to select prototypes for instance-based classifiers whose predictions are to be combined.

Little research has been done on combining nearest neighbor classifiers through any means. In fact, as we shall see in the next chapter, two research efforts that have attempted to combine nearest neighbor classifiers have concluded that they are *not* suitable for combination, at least not within the frameworks studied. One of the results stemming from the investigation of this dissertation's hypothesis is that this negative conclusion is not warranted for all varieties of nearest neighbor classifiers. We create nearest neighbor classifiers that can be combined effectively on a variety of data sets.

In one set of experiments, Breiman [1992] demonstrated that combining 100 nearest neighbor classifiers did not reduce the rate of misclassification below that of a single nearest neighbor classifier. He attributed this behavior to a characteristic of nearest neighbor classifiers: that the addition or removal of a small number of training instances does not change nearest neighbor classification boundaries very much. Decision trees and neural networks, he argued, are in this sense less *stable* than nearest neighbor classifiers.

One of the key ideas of this thesis is that nearest neighbor classifiers may be made more suitable for combination if a measure of *instability* is deliberately injected into them. In Breiman's experiments, the component nearest neighbor classifiers stored a large number of prototypes. We introduce instability by storing a very small number of prototypes in each classifier. When only a small number of prototypes are stored, the classifiers become less stable: small changes to the set of stored prototypes will make relatively large changes to a concept hypothesis.

Stability is a desirable characteristic in general for classifiers. Stable classifiers are robust, since they do not make radical changes to a concept hypothesis in response to minor training set changes. However, stability may be a less desirable characteristic of component classifiers whose predictions are to be combined.

This dissertation is a step in a research program whose ultimate goal is to understand how to identify and use prototypes and other salient examples for classification and for other tasks, particularly for tasks that require diverse reasoning elements to be combined [Rissland and Skalak, 1991; Skalak, 1992; Skalak and Rissland, 1992; Skalak, 1993; Skalak, 1994;

Rissland *et al.*, 1996]. Example tasks that have been investigated include classification, rule interpretation, hybrid reasoning, legal argument and information retrieval. In this dissertation, we focus on classification.

Our problem is at the confluence of two streams of research: nearest neighbor classifier configuration and composite classifier construction. Until now, these issues have been treated independently. One side-effect of this research will be methodological. Our research demonstrates that one path to progress on both problems is to treat them as dependent problems to be solved simultaneously. Constructing composite classifiers using nearest neighbor building blocks can suggest ways to configure nearest neighbor classifiers — by selecting prototypes, in particular — and exploring the ways to configure nearest neighbor classifiers can offer new classifiers to combine.

1.2 Approach to the Problem

Our review of research to date on classifier combination will reveal three important design goals: (1) component accuracy, (2) component diversity and (3) component efficiency. We discuss these criteria in detail in the next chapter. In short, they translate into using component classifiers that independently classify well, classify differently and classify quickly.

There is tension among these competing goals. In particular, it is difficult to a create classifier that is both accurate and diverse. It is difficult as well to create a classifier that is both accurate and efficient. Previous research has resolved these tensions — almost always implicitly — in favor of component accuracy. Our perspective, adopted in this dissertation, is to elevate the importance of *diversity* and *efficiency*. We begin by choosing a model class that we expect to provide a balance of these three goals. We use homogeneous nearest neighbor classifiers that store only a very small number of prototypes per class, often just one prototype per class. We refer to these classifiers as "simple", "small" or "minimal" (if only one prototype is stored), or as creating "coarse hypotheses", depending on the aspect under scrutiny. In Chapter 3, we show that these classifiers can be highly accurate on a variety of data sets.

Diversity. Component diversity is an elusive, but necessary, factor in classifier combination. Where the components make exactly the same predictions, there is no hope that combining those predictions will improve accuracy. Only where the component classifiers make different errors — and make them with a regularity that can be learned by a combining classifier — will we have the potential to improve results through combination. In part, we gain diversity through our model class choice of minimal nearest neighbor classifiers. We also propose other, explicit methods, described in Chapter 4, which manipulate a training set to produce diverse classifiers.

Efficiency. Reliance on minimal nearest neighbor classifiers is an important characteristic of our work, and so we take some care to outline why we focus on this little-explored set of classifiers. Even the voluminous research on editing algorithms for prototype selection typically deals with classifiers that store a much larger percentage of the training set (Chapters 2 and 3). To explain our emphasis on minimal nearest neighbor classifiers, we discuss the choice of this model class from two perspectives. First, we explain the choice in terms of reasons to adopt classifiers that create *simple hypotheses*. Second, we defend the choice of this model class as a means to lower *computational expense*.

Simple Hypotheses. We give four reasons why simple hypotheses are worthy of extensive investigation.

First, by adopting a "simplicity first" methodology, we will be less likely to overlook simpler solutions to our classification tasks [Holte, 1993]. This general preference is expressed by Occam's Razor, a philosophically controversial contention. Occam's Razor is the hypothesis that, all other things being equal, simpler explanations are to be preferred over more complex ones [Kemeny, 1953; Blumer *et al.*, 1987]. In our classification setting, it boils down to a preference for simpler classifiers over more complex ones. A simpler classifier is one that generates simpler concept hypotheses, which are shorter in some concept representation language [Angluin, 1992]. For nearest neighbor classifiers, where a concept hypothesis is defined in terms of a set of prototypes [Freund and Schapire, 1996], storing fewer prototypes would entail shorter concept hypotheses and fewer decision regions.

A second reason to prefer simpler classifiers is to assuage the problem of overfitting. Where overly complex hypotheses are permitted, there is a potential for overfitting a training set. Due to the many degrees of freedom present in a typical composite classifier, it is especially important to guard against overfitting the training set [Brodley, 1994]. While some data sets may be overfit even with simple classifiers, we mitigate this potential problem by applying nearest neighbor classifiers that store few prototypes. The danger of overfitting is still present when combining simple classifiers, but may well be less severe, because simple classifiers admit concept hypotheses that are less likely to be overly complex and because they admit fewer hypotheses.

A third reason to investigate simple nearest neighbor classifiers can be traced to a demonstration of the utility of simple, shallow decision trees, sometimes called *decision stumps* [Holte, 1993]. Holte's research showed that a very simple classification rule is sufficient to perform quite good classification on commonly used databases. Specifically, it was shown that one-layer decision trees that classify on the basis of a single attribute can be surprisingly accurate on a number of tasks drawn from the U.C.I. Machine Learning Repository [Murphy and Aha, 1994]. Our results complement that work by showing that small nearest neighbor classifiers work well on a variety of U.C.I. data sets. Holte's work is controversial, however, and his results may well reflect more about the data used than about the model class applied. Utgoff and Clouse have recently called into question the utility of smaller decision trees, based on their experiments with a little-known metric for decision-tree induction, the Kolmogorov-Smirnov metric [1996].

Fourth, simpler hypotheses are usually more comprehensible, an important consideration in data-mining applications, for example [Cherkauer and Shavlik, 1996]. The general utility of simple classifiers in data mining, or knowledge-discovery systems, provides an additional reason that further research is warranted on classifiers that generate coarse hypotheses. Typically, in knowledge-discovery systems, simple rules or shallow decision trees are extracted from a large database. The ability to build understandable models is paramount in these applications, and has been shown to be of immediate and substantial economic benefit.

Computational Expense. Thus, there are many reasons to use simple classifiers, which yield efficiency as a side-effect of a bias in favor of simplicity. However, computational expense is still a pressing consideration, independent of simplicity. Even as computers become faster and have more memory, we tend to use them to solve larger and larger problems. It still would be impossible to create a nearest neighbor classifier that stores a million-record database entirely in the primary memory of most desktop computers. Efficiency would also be important for composite classifiers that are to be embedded as real-time, on-line learning components in a dynamic agent, such as a robot.

Further, efficiency can assume larger importance in composite architectures, which consist of component classifiers that must be trained and applied separately and sometimes serially. Where, for example, a large number of backpropagation networks are combined, training costs can be enormous. Even the proponents of a successful classifier combination technique, called *error-correction output coding* (*ECOC*, described in Chapter 2), which can require combining several hundred classifiers, have admitted that attaining an efficient level-1 representation is still "[a]n important open problem" [Kong and Dietterich, 1995, p.321] despite the utility of ECOC. Further, there are techniques that have been proposed for combination that are evidently computationally infeasible today for many tasks, the power of scientific workstations notwithstanding. One example is "bagging" [Breiman, 1994a] plus ECOC, which potentially can require the training of tens of thousands of neural networks. Bagging is also discussed in Chapter 2.

Finally, the model class that we have adopted, nearest neighbor algorithms, generally has been recognized as inefficient to apply. The standard nearest neighbor algorithm computes the similarity between each stored prototype and a test instance [Cover and Hart, 1967; Dasarathy, 1991], and this computation can entail significant costs, especially where many prototypes with many features are present. A standard nearest neighbor algorithm computes the similarity of a test instance to each stored prototype, and is therefore $O(n)$, where n

is the number of prototypes [Dasarathy, 1991]. Adopting the constraint that each classifier store a small number of prototypes is an extreme version of the classical response to the nearest neighbor inefficiency — prototype selection. While this distinction between a standard nearest neighbor classifier and a simple one is only a matter of the number of prototypes, this distinction will make a difference in the utility of their combination.

We take a straightforward approach to efficiency in this dissertation by adopting two constraints: (a) simple components and, in addition, (b) few components. On a typical serial workstation, where only a few component classifiers have to be trained and applied, the computational expense will be lower than with more components. As we shall see, the constraint of few components is reasonable in light of previous research showing that the incremental benefits of combination often evaporate after adding a fairly small number of components. (If it were possible to create classifiers that made independent errors, this evaporation would not be seen, however.)

As we have noted, it is necessary that the component classifiers make diverse errors: if they made the same errors, no benefit would accrue from combining them. The design goals of diversity and efficiency are like twin binocular focusing knobs to bring into clearer view the central hypothesis of this thesis, which we can now refine and restate.

The accuracy of a standard nearest neighbor classifier can be improved or exceeded through composite-classifier architectures that incorporate a small number of diverse component nearest neighbor classifiers, each of which stores only a small number of prototypes.

As we shall see in Chapter 2, this dissertation investigates a different part of the conceptual space of composite classifiers from previous research (Figure 1.2). Composite classification research may be characterized partly by two dimensions: (1) the granularity of the hypothesis space of each component classifier, and (2) the number of component classifiers that are combined. The granularity of the hypothesis space may be characterized qualitatively as running from coarse to fine. An example of a coarse hypothesis space is the set of hypotheses generated by a nearest neighbor classifier that stores a very small number of prototypes. An example of a fine hypothesis space is the set of hypotheses generated by a multilayer neural network with a large number of units. The number of component classifiers may be characterized qualitatively as varying from few to many. For this purpose, two or three may be considered few; up to ten, a moderate number; and ten or more, such as 100, may be considered many. Implementations of such algorithms as ECOC [Kong and Dietterich, 1995], bagging [Breiman, 1994a] and Adaboosting [Freund and Schapire, 1996], which have proven capable of producing more accurate composite classifiers (except when combining nearest neighbor components), typically have combined a large or a moderate number of fine-hypothesis classifiers. These algorithms are discussed in Chapter 2. By contrast, this work focuses on combining a small number of coarse-hypothesis classifiers.

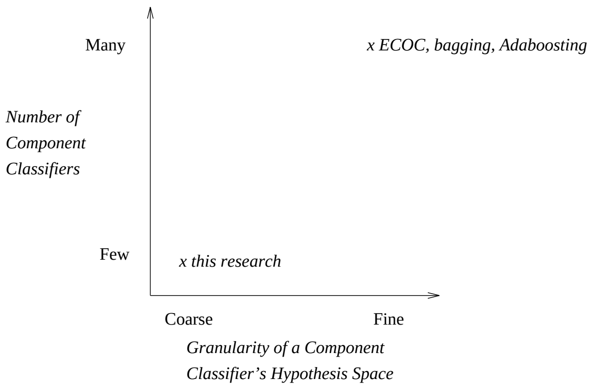

Figure 1.2: Position of this research and related algorithms with respect to the dimensions of the number of component classifiers combined and the granularity of the hypothesis space of each component.

Adopting these design constraints actually yields some surprising results. For example, we give a simple algorithm that creates a nearest neighbor component classifier that stores what is evidently an absolute minimum number of prototypes, one per class. On a clear majority of the standard data sets tested, this classifier performs as well as or better than a nearest neighbor classifier that stores all training instances as prototypes.

There is a also a procedural side-effect to adopting these constraints. These constraints allow us to work on a simplified version of the general classifier combination problem. Simplification permits us to observe a working adage of mathematicians: "Work on the simplest form of the problem that you don't understand completely." Investigating a simple form of the problem can clarify general principles that then can be applied to the problem in its more intricate guises.

1.3 Overview of the Results

This dissertation studies the general problem of classifier combination by focusing on the specific problem of combining nearest neighbor classifiers. Our work represents a substantial advance along several distinct, but complementary, research dimensions. Below, we describe some of the main contributions of the thesis.

1. **Prototype selection.** The research presented in this dissertation extends previous work on prototype selection for an independent nearest neighbor classifier. We show that in many domains only a very small number of prototypes can provide classification accuracy equal to that of a nearest neighbor classifier that stores all training instances as prototypes. We extend previous work by demonstrating that algorithms that rely primarily on random sampling can effectively choose a small number of prototypes.

2. **Composite nearest neighbor classifiers.** We undertake the first study that investigates in depth the combination of nearest neighbor classifiers. Although previous research has questioned the utility of combining nearest neighbor classifiers, we introduce algorithms that in many domains create a stacked generalizer that improves classification accuracy over the component nearest neighbor classifiers and is more accurate than a nearest neighbor classifier that uses all training instances as prototypes.

   We investigate the relationship between the accuracy and the diversity of the component classifiers in a stacked classifier. Our experiments show that the relationship of these two design criteria is complex and depends to a large extent on the data. In particular, we demonstrate that stacked generalizers with the highest test accuracy do not necessarily demonstrate maximal degrees of component training accuracy and diversity.

1.4 Guide to the Dissertation

Rhetorically, the dissertation follows a canonical outline. The important questions we try to answer have been posed in this introduction. We have also discussed the constraints on our approach to the task, and pointed to some of the results that we will attain. In the next chapter, we survey relevant work to provide a historical context for our investigation and to describe the research upon which we will draw. After surveying related work, we introduce our approach to the underlying problems and distinguish it from other approaches. In the body of the dissertation, we evaluate our methods experimentally, analyze the results and draw conclusions. Finally, we summarize the dissertation and point to directions for additional research.

Specifically, the dissertation is organized into the following chapters:

Composite Classifier Construction (Chapter 2). We analyze the design criteria for composite classifiers and review three of the main composite classifier architectures: boosting, stacked generalization and recursive partitioning. We present a taxonomy of methods for creating diverse classifiers to be used as components and discuss research to date implementing those methods. In particular, we describe the few attempts at creating composite classifiers using nearest neighbor components, and survey approaches to the prototype selection problem for a single, independent nearest neighbor classifier.

Nearest Neighbor Composite Classifiers (Chapter 3). We describe our approach to nearest neighbor classifier construction, in particular to the construction of nearest neighbor classifiers that are to be incorporated as components in a stacked generalizer. A sampling algorithm is introduced for choosing prototypes to be used in a nearest neighbor classifier. In this chapter we present a new taxonomy of instance types, which is based on statistics gathered from applying a sample of classifiers to each instance.

Boosting (Chapter 4). We attack a specific version of the general boosting problem, which is the problem of increasing the accuracy of a given classifier. We present four new algorithms for boosting a nearest neighbor classifier that stores all training instances as prototypes.

These algorithms invoke either a new method for creating component diversity or extend an existing method to nearest neighbor classifiers. We provide a comparison of these methods for creating component classifiers that make different errors.

Stacked Generalization (Chapter 5). We investigate a two-layer stacked generalizer, consisting of nearest neighbor component classifiers and a combining classifier. Several combining algorithms are compared, and we show that voting is not always the best combiner. We show that the relationship between the fundamental criteria of accuracy and diversity of the component classifiers varies widely from data set to data set. We also introduce a generalization of the traditional k-nearest neighbor classifier, which arises from stacking *jth-nearest neighbor* classifiers. Finally, we present an algorithm that improves composite classifier performance by defaulting to the prediction of the most accurate component for instances that are likely to be misclassified by the composite classifier.

Conclusions (Chapter 6). We summarize the contributions of this thesis to composite classifier construction and to machine learning. We suggest additional questions for research that are raised by our methods and our results.

Table 1.1 summarizes the topics covered in the three primary areas: prototype sampling (Chapter 3), boosting (Chapter 4) and stacked generalization (Chapter 5).

Table 1.1: Topic summary table.

Area	*Topics*	
Prototype Sampling	Accuracy	Number of Samples
	Clustering	Instance Taxonomy
Boosting	Coarse Reclassification	Deliberate Misclassification
	Composite Fitness	Composite Fitness–Feature Selection
	Accuracy-Diversity	Failure Analysis
	Selective Superiority	Schapire's Boosting
	Safe-Coarse Reclassification	
Stacked Generalization	Combining Functions	Bootstrap Components
	Number of Components	Accuracy-Diversity
	Probability Estimates	k-NN Generalization

The assumptions we make about the configurations of the component k-nearest neighbor classifiers for boosting and stacked generalization are given in Table 1.2. In this thesis, boosting combines two component classifiers C_0 and C_1, where C_0 stores all training instances as prototypes and C_1 stores one prototype per class. Stacked generalizers have two or more component classifiers C_i, where each C_i stores one prototype per class. All k-nearest neighbor classifiers use $k = 1$ and apply the distance metric defined in Chapter 3.

Table 1.2: Component classifier summary table: prototypes stored in component classifiers for boosting and stacked generalation in this thesis.

	Boosting *2 Components*	*Stacked Generalization* ≥ 2 *Components*
C_0 *prototypes*	all training instances	one per class
C_1 *prototypes*	one per class	one per class
C_i *prototypes*, $i > 1$		one per class

CHAPTER 2

COMPOSITE CLASSIFIER CONSTRUCTION

In this chapter we introduce the general problem of classifier combination and discuss related research. The goal of the chapter is to give the reader a context for the problem of selecting prototypes for nearest neighbor classifiers whose predictions are to be combined.

The chapter is structured from general to specific. First, we discuss the design criteria that govern composite classifier design. Second, we describe several composite architectures that satisfy one or more of the design goals. Third, we look separately at two parts of a composite classifier: the component classifiers and the combining classifier. To organize the research on these two parts, we give taxonomies of the learning algorithms that have been used. Fourth, we review research on prototype selection for a single, independent nearest neighbor classifier. The chapter thus begins with the most general considerations that govern composite classifier design and descends to the selection of the specific instances to store in a component nearest neighbor classifier.

2.1 Composite Classifier Design Criteria

There are three primary criteria that have been applied to creating composite classifiers:

1. **Accuracy** of the component classifiers,
2. **Diversity** of the component classifiers, and
3. **Efficiency** of the entire composite classifier.

The accuracy criterion arises from a desire to make the component classifiers independently accurate. By "independently," we mean when applied alone as a single classifier.

The diversity criterion arises from the simple observation that combining the predictions of a set of classifiers that all make the same errors cannot lead to any improvement in the accuracy of the composite prediction.

The efficiency criterion is less often considered in composite classifier work, but arises from the general requirement that a classifier should use only reasonable amounts of time and memory for training and application.

Only recently has attention been focussed directly on these criteria, and much past research only implicitly has taken account of them. Only a miniscule amount of research has been done to determine the relationships among these criteria. We consider the findings of previous research with respect to each criterion, individually.

While the criteria cannot be divorced totally from the type of composite architecture, we focus on the stacked generalization framework discussed in Chapter 1.

2.1.1 Accuracy

The accuracy of the component classifiers has been regarded as the most important of the three criteria, and a great deal of effort typically goes to training the individual components to be highly accurate as independent classifiers. If the predictions that are being combined are not accurate, this intuition goes, then the ultimate prediction cannot possibly be highly accurate.

For example, the default way to combine the predictions of a set of classifiers is voting. Where voting is used as the combiner, it can be seen that combining accurate classifiers may be important. Inaccurate classifiers cannot then be used to improve accuracy. From experiments with voting and several other combining algorithms, Ali and Pazzani conclude "To achieve higher accuracy the models should be diverse *and* each model must be quite accurate." [1996, p.124][emphasis in original]. We will return to this observation when we examine the requirement of diversity.

By contrast, we show that component accuracy may not be of paramount importance in classifier combination. In general, the role of the component classifiers in a stacked generalizer is to provide features for the combining classifier. As long as the combining algorithm has a learning bias that allows it to generalize from those features, we argue that it is not necessary that the component classifiers be independently accurate. This observation gives us the design flexibility potentially to sacrifice component accuracy in favor of increasing diversity. For example, Brodley's general, recursive system for algorithm selection, Model Class Selection (MCS), in some cases will not choose the most accurate component classifier for a set of instances. Under some circumstances, the MCS algorithm prefers a classifier that makes the resulting subspace easier to learn over one that classifies the available subspace most accurately [1994].

It is obvious that when a composite classifier's accuracy is higher than the most accurate component, the composite classifier has been created by combining the most accurate component with classifiers that are *less* accurate than the best component (unless they all happen to be of equal accuracy). In Section 2.2.1.1, we will study an example in which a classifier that was 69% accurate on a word pronunciation task was boosted to 88% accuracy by combining its predictions with two classifiers that were 23% and 25% accurate when applied individually [Wolpert, 1992]. This result provides a dramatic illustration that component classifier accuracy is not always the most important factor.

2.1.2 Diversity

A number of researchers have observed, with varying degrees of formality, that classifiers must make different errors in order to achieve higher accuracy when combined. For example, Breiman observed that the "biggest gains came when dissimilar sets of predictors were stacked" [1992, p.4]. Wolpert has ventured "one should try to find generalizers which behave very differently from one another, which are in some sense 'orthogonal,' so that their guesses [predictions] are not synchronized" [1993, p.6]. Battiti and Colla note that a "sufficiently large uncorrelation in . . . mistakes" [1994, p.691] leads to boosted classification accuracy. In their work on error-correcting output codes, which we describe in Section 2.2.1.2, Kong and Dietterich observe an improvement in accuracy from combining a large number of classifiers when the component classifiers are not highly correlated [1995].

Hansen and Salamon have demonstrated that assuming that the errors made by a set of neural networks are independent allows the network predictions to be modeled using a binomial distribution [1990]. They show that if the error rate of each network is less than 50%, then adding more networks that make independent errors will monotonically decrease the overall error rate, where majority vote is used to combine the outputs of the networks. To see this fact, if each network has error rate p , then the probability of k errors among N independent networks is given by the binomial distribution:

$$\binom{N}{k} p^k (1-p)^{N-k}$$

.

It is then easy to compute the probability that a majority vote of the networks is incorrect [Hansen and Salamon, 1990, p.995]. Hansen and Salamon acknowledge an important point, however: that networks rarely make independent errors.

Ali and Pazzani have shown that error is most reduced by using component classifiers whose errors are negatively correlated [1996]. They observe a linear relation between a measure of error reduction and the degree to which the component classifiers make correlated errors. However, our own work will build on the comment by Ali and Pazzani that "To achieve higher accuracy the models should be diverse *and* each model must be quite accurate. In fact, it is easy to produce uncorrelated errors by learning less accurate models." [1996, p.124][emphasis in original].

The reason that classifier combination is difficult is that it is hard to satisfy the requirements of diversity and accuracy simultaneously. Our research plan (developed before the publication of Ali and Pazzani's observation, incidentally [Skalak, 1995]), may be seen as investigating how Ali and Pazzani's second speculation — that it is easy to produce uncorrelated errors if the accuracy constraint is relaxed — might provide a key to one approach to classifier combination. Our first approach to relaxing the accuracy constraint is to choose a model class that is likely to underfit many data sets.

Opitz and Shavlik present the only work found that explicitly attempts to deal with the relationship between accuracy and diversity [1995]. They apply a genetic algorithm to the task of architecture selection for a composite classifier that combines a set of neural networks. The fitness function that they apply maximizes a linear combination of classifier accuracy and diversity, but the relative weights are permitted to vary, according to whether the ensemble error and the population diversity are increasing or decreasing. From examining the relative trend of error and diversity, heuristics determine whether to change the relative weighting. The initial weighting heavily favors accuracy.

2.1.2.1 Measuring Diversity

At least two approaches to measuring diversity have been implemented.

1. The Ali and Pazzani [1996] measure compares the output of each of the components with each other and with the (correct) target class.

2. The Opitz and Shavlik [1995] measure compares the output of a component with the output of the entire composite classifier. This measure has the advantage that the class target for each instance need not be known, so that it may be useful for an unsupervised learning task as well.

The measure of diversity used by Ali and Pazzani is appropriate for classification problems, rather than for function approximation problems, since it relies on an exact match of component predictions. To use their notation, we assume we have been given a target function f (whose range is the set of class labels) and a collection of models (component classifiers) $\mathcal{F} = \{\hat{f}_1, \hat{f}_2, \dots, \hat{f}_n\}$. Let $f(x) = S_i$ denote that instance x belongs to class S_i. Similarly, $\hat{f}_j(x) = S_i$ means that the model $\hat{f}_j$ predicts class S_i for instance x.

The probability that models $\hat{f}_i$ and $\hat{f}_j$ make the same error is given by

$$p(\hat{f}_i(x) = \hat{f}_j(x), \hat{f}_i(x) \neq f(x)).$$

So the degree of error correlation between the models in $\mathcal{F}$ is defined as

$$\phi_e(\mathcal{F}) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} p(\hat{f}_i(x) = \hat{f}_j(x), \hat{f}_i(x) \neq f(x)).$$

This definition of error correlation is thus the average probability that two models make the same error, taken over all pairs of models.

Opitz and Shavlik adopt a measure of diversity that was previously investigated by Krogh and Vedelsby [1995]. This measure is designed in particular for neural networks that output a real value. Placing the measure into the terms used by Ali and Pazzani, the diversity of the network (model) i is defined as

$$D_i = \sum_{x \in T} [\hat{f}_i(x) - \bar{f}(x)]^2,$$

where T is the training set and $\bar{f}$ is the output of the entire composite classifier, in which the output is a weighted average of the component networks,

$$\bar{f} = \sum_{i=1}^{n} w_i \cdot \hat{f}_i,$$

where $\sum_{i=1}^{n} w_i = 1$.

Thus, diversity is measured in terms of the difference between the output of the composite and the output of a component. (It differs from the standard mean-squared error in that D_i does not take the true value, $f(x)$, into account.)

Given a measure of diversity, a learning algorithm may search for a set of diverse models, by using the diversity as an evaluation function. The difficult questions remain — how to make the search efficient and effective.

2.1.3 *Efficiency*

Avoiding prohibitively expensive classifiers has been cited previously as a design goal [Brodley, 1994]. As we discussed in Chapter 1, all other things being equal, one ought to prefer (a) fewer component classifiers over more, and (b) computationally inexpensive ones over more expensive.

2.1.3.1 Number of Component Classifiers

Clearly, given equal performance, one would prefer fewer component classifiers, since training and application costs will be lower than with more components. Since composite classification requires training a number of component classifiers and a combining classifier, there is a strong impetus to limit the number of the component classifiers in a composite system.

The question has not been closely examined by previous research, but there have been several indications in the literature that a maximal degree of composite accuracy fortuitously can be reached with the use of a fairly small number of classifiers. Working on a handwritten digit recognition task, Battiti and Colla observed that only two to three neural networks sufficed to give significantly higher accuracy than the best they were able to obtain from an individual net [1994]. Studying a weather prediction task and a computer science student course prediction task, Kwok and Carter's experiments showed that the error rate reached a minimum with only a small number of component decision trees (three or fewer), and that a slight degradation was evident as more trees were added [1990]. Schapire's original boosting algorithm, discussed in Section 2.2.2, requires only the use of three component classifiers, although it can be applied recursively. In his experimental evaluation of stacking various regression algorithms, Breiman found that only approximately three regression algorithm outputs (out of 40) contributed to the ultimate stacked prediction on average [Breiman, 1992]. Perrone found that the maximum accuracy occurred when only six to eight networks were combined by weighted averaging, and that the most striking increase in accuracy occurred when two networks were combined [1993]. Mani argues for combining a small number of neural networks, by analogy to the small number of stocks required to diversify a portfolio [1991].

Collecting these previous experiments reveals that a very small number of classifiers sufficed to give maximal or near-maximal accuracy for a variety of tasks. This previous work suggests that it will be useful to investigate even the limiting case, where only two component classifiers are created.

Additional support for a small number of component classifiers comes from different source — psychological studies of human perception. In particular, the human visual system apparently performs as though the "stimuli [exist] in a low-dimensional metric psychological space" [Intrator and Edelman, 1996, p.4]. Intrator and Edelman use this observation and other evidence from experiments with human vision to argue that a low-dimensional representation supports the *transfer* of skill across tasks. For our composite classifiers, fewer component classifiers entails a smaller number of features in the level-1 representation.

2.1.3.2 Computational Resources of a Component Classifier

Although training costs have sometimes been very high, there has been little research attention paid to the specific problem of promoting efficiency in component classifiers. In particular, where large, multilayer neural networks are trained according to some learning rules, the cost of training can be great. One example that we shall encounter is that the error-correcting output codes often relied upon require hundreds of component classifiers (Section 2.2.1.2). Even a composite classifier that consists of a linear combination of ten multilayer networks trained with backpropagation (as by Perrone [1993]) can be computationally very expensive to train. Excessive costs also have the detrimental methodological side-effect of slowing the pace of experimentation.

Drucker points out an additional benefit of building fast classifiers: they can be combined with slow ones [1996]. On a handwriting recognition problem, at classification time, he found that an ensemble of trees was much faster than a neural network. Speed and accuracy in this application were then increased by using the ensemble as a pre-classifier, and used the slower, but more accurate, neural classifier only for the examples for which the pre-classifier was unsure. Thus, a fast classifier can be useful for this serial combination.

2.2 Composite Classifier Architectures

In this section we move from general design considerations to the methods that previous researchers have used to attain them. Classifier combination is known under a number of names, depending on the research community and the application, including *ensemble* or *consensus methods*, *hybrid* or *composite models*, *fusing*, *estimator combination* and *forecast combination*, *aggregation* or *synthesis*. Here we have adopted the terminology that in a *composite classifier* the predictions of *component classifiers* are amalgamated by a *combining classifier*.

Of course, there are many architectures for classifier combination. The three primary architectures for combining classification algorithms are:

1. **Stacked Generalization**
2. **Boosting**
3. **Recursive Partitioning**

Described in detail below, the three architectures are similar in that they try to reduce generalization error by combining the predictions of component classifiers.

In brief, *stacked generalization* is a recursive layered framework for classifier combination in which the layer of classifiers at each level is used to combine the predictions of the classifiers at the level immediately below. *Boosting* is an attempt to increase the generalization accuracy of a given classifier. It successively creates complementary component classifiers by filtering the training set. To give an ultimate prediction, a vote is taken of the predictions of the given classifier and the newly created components. *Recursive partitioning* is a method for classifier combination in which the domain space is recursively sub-divided into regions where particular classifiers have jurisdiction to make a prediction.

No meta-generalizing scheme is guaranteed to yield a classifier with a minimal generalization error [Wolpert, 1992]. This inevitable result is due to the *ill-posed* nature of the problem of trying to make predictions for a (characteristic) function, given only a finite set of examples.

Working within these theoretical limitations, we shall try to see what success can be attained with particular architectures on commonly used data. For example, on a difficult protein folding prediction task, stacked generalization using a probabilistic classifier, a memory-based classifier and a neural network attained accuracy not achieved by any other learning algorithm [Zhang *et al.*, 1992; Wolpert, 1993].

2.2.1 Stacked Generalization

Much of this dissertation is an outgrowth of research on the *stacked generalization* framework for combining classifiers [Wolpert, 1992; Wolpert, 1993]. Wolpert concentrates on a two-layer architecture in which the classifiers to be combined are called *level-0* classifiers, and the combining classifier is the *level-1* classifier. The layering may be iterated to create level-2 classifiers, and so on. The idea is not original to Wolpert of layering classifiers that themselves could be applied on a standalone basis, however. The idea can be traced at least as far back as Nilsson's committee machines [1990] (originally published in 1965) and Selfridge's Pandemonium for pattern recognition, whose architecture is very similar to the standard two-layer stacked generalizer as shown in Figure 1.1 [Selfridge, 1959]. Pandemonium was a winner-take-all system in which only the prediction of the agent with the loudest shout (the greatest level of confidence) was applied. Pandemonium did not attempt to combine the predictions of the pattern-recognizing agents, however.

Stacked generalization is a framework for classifier combination in which each layer of classifiers is used to combine the predictions of the classifiers at the immediately preceding layer. A single classifier at the top-most level outputs the ultimate prediction. The classifier at each layer receives as input a vector of predictions of the classifiers in the layer immediately below. While the information passed from layer to layer may take the form of vectors of predictions, confidence values, or other data, we will limit our attention to systems in which only class predictions are passed from layer to layer. This is the usual assumption, although the stacked generalization framework is not limited to this choice. We will also limit ourselves to two-layer stacked generalizers, consisting of a set of component classifiers and a single combining classifier that combines the predictions of the component classifiers.

The terminology for the levels of classifiers is somewhat confusing, since it conflicts with the usual data structure terminology for trees. Research on stacked generalization refers to the classifiers at the bottom level as *level-0* classifiers. The classifiers at the next level, which combine the predictions of the level-0 classifiers, are called the *level-1* classifiers. This terminology is inconsistent with the standard terminology for a tree, where the root node is referred to as "level-0" and the leaf nodes are at the highest-numbered level of the tree. Unfortunately, the level-0 and level-1 terminology is entrenched. Often we will refer to component and combining classifiers to avoid confusion.

In effect, stacked generalization is an attempt to minimize generalization error by using the classifiers in higher-numbered layers to learn the types of errors made by the classifiers immediately below. In this way, it is related to statistical methods for classifier selection such as *cross-validation* and *bootstrapping*, as means for minimizing generalization error. Cross-validation partitions a training set into n subsets. For each of n plies, a different subset is held out as a test set, and the union of the remaining $n-1$ subsets constitutes the training set. Generalization accuracy on the withheld test set is computed for each ply, and the average over the n plies is taken. Bootstrapping is similar to cross-validation, except that rather than partitioning the data set into n disjoint subsets, training and test sets are created by random sampling of the original data set with replacement [Efron, 1979].

With cross-validation and bootstrapping, the "combining" mechanism is "winner take all." The single classifier with the lowest cross-validation error is selected. The theory of stacked generalization is that there may be a more intelligent way to use a set of contestant classifiers. Rather than select the apparently most accurate one, combine the predictions of the contestants. The task of the level-1 (and higher) classifiers is to learn to use the contestant predictions to predict more accurately.

Networks of nearest neighbor computational units. The stacked generalization framework diagram looks like a multilayer neural network diagram (Figure 2.1). There are certainly analogous aspects to the two frameworks. The distinction between them appears to lie partially in the type of information that is passed from the input layer to the succeeding layer and in the granularity of the classifier nodes themselves. In a neural network, an activation value is passed to forward layers, which may or may not be an ultimate prediction or even have some recognizable interpretation. Generally, in the stacked generalization framework, a "full-fledged" class prediction is passed to the combining classifier, and not just a scalar that somehow contributes to a prediction [1]. Also, in other implementations of stacked classifiers, the classifiers to be stacked are complex, and may be neural networks themselves [Perrone, 1993].

So one view of this research is as presenting a framework for combining classifiers that is an intermediate point between (a) complex, computationally expensive component classifiers that are designed to be accurate on their own and (b) neural network computational units that are very simple and are designed to be applied together in architectures that often apply many such units.

In some previous work this intermediate point has been struck, in which classifiers can be applied as simple classifiers in their own right or in networks. For example, linear threshold units (LTU's) could be used in layered feed-forward networks or in perceptron trees [Utgoff, 1989], or they could be used as classifiers on their own. By analogy, this proposal advocates combining small instance-based classifiers that could be called "nearest neighbor units". Nearest neighbor units can be used on their own, as we show in Chapter 3, but can also be combined into layered networks, as we show in Chapter 5. This perspective places the current work at one of a few intermediate points on the spectrum from neural networks that apply very simple elements to composite systems that integrate a small number of large-grained modules.

2.2.1.1 Architecture and Algorithm

The idea of stacked generalization is quite simple, but Wolpert was apparently the first to discuss the idea in its full generality. As characterized by Wolpert, stacked generalization assumes that we have been given a set of n level-0 (component) learning algorithms, a level-1 learning (combining) algorithm, and a training set of classified instances, T_0. The n level-0 learning algorithms should be distinct, so that diverse level-0 classifiers are obtained. Otherwise, no synergy will result from their combination. How to create diverse component classifiers

[1] However, the application of a squashing function may make the output appear binary, and therefore more like a class prediction.

is a fundamental problem for composite classifier construction. We survey and taxonomize approaches to the diversity problem in Section 2.3.

Although the original description of stacked generalization is expressed in terms of learning real-valued functions with real-valued input features, the framework is immediately extensible to symbolic attributes and outputs. The algorithm has the usual two phases, training and application. Pseudocode for training and applying a two-level stacked generalizer is given in Figure 2.2.

Training Phase:

1. Train the component classifiers using leave-one-out cross validation as follows. For each instance in the data set, train each of the n level-0 classifiers using the remaining instances. After training, classify the held-out instance using each of the trained level-0 classifiers. Form a vector from the predictions of each of the level-0 classifiers and the actual class of that instance. As shown in Figure 2.2, these vectors have length $n + 1$, since they have as components the predictions of each of the n level-0 component classifiers and a class label.

2. Train the level-1 classifier, using as the level-1 training set the collection of vectors of the level-0 classifier predictions and the actual classes. This collection has cardinality $|T_0|$, since there is one level-1 training instance corresponding to each level-0 training instance (Figure 2.2).

3. Since the level-0 classifiers have not been trained on the entire training set, re-train the level-0 classifiers on the entire training set.

Application Phase: When presented with a new instance whose class is unknown, classify the instance using each of the level-0 classifiers, deriving an input vector for the level-1 classifier. The derived vector is then classified by the level-1 classifier, which outputs a prediction for the new instance.

Leave-one-out cross validation is applied in the training phase to ensure that the level-1 algorithm is trained on the generalizations made for unseen data by the level-0 classifiers. Since "generalization" refers to data outside the training set, this observation is memorialized in the name "stacked generalization," as opposed to "stacked classification". In an experiment with combining linear regression functions, Breiman showed that using 10-fold cross validation to create the level-1 training data yielded slightly more accurate stacked generalizers than leave-one-out cross validation [1992].

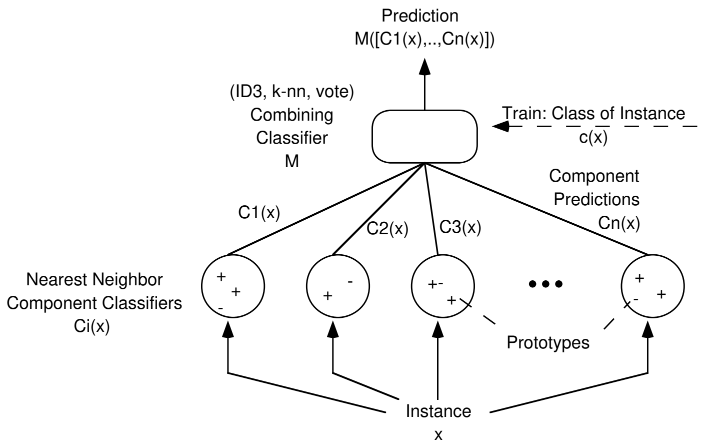

Figure 2.1: Stacked nearest neighbor classifier architecture. In the training phase, the predictions of each component classifier and the actual instance class are supplied to the combining classifier.

Wolpert has applied the stacked generalization framework to the NetTalk problem of phoneme prediction from letter sequences. The experiment was designed to test "whether stacked generalization can be used to combine separate pieces of incomplete input information" [Wolpert, 1992, p.253] using a set of three so-called "HERBIE" level-0 classifiers combined by a level-1 HERBIE classifier. Wolpert's HERBIE classifiers are 4-nearest neighbor classifiers that use a weighted Euclidean distance metric when applied to instances with real-valued features and that use a Hamming metric for symbolic features. (When the Euclidean metric is applied, the weighting factor for each neighbor is the reciprocal of the distance between that neighbor and the test instance.)

Seven-letter windows were passed to the three level-0 classifiers, but the three HERBIEs received as input just the third, fourth, and fifth letters in the window, respectively. In other words, each classifier had a one-dimensional input space, and differed only in the letter slot

Key:

T_0 : Training set in original (level-0) representation

T_1 : Training set in level-1 representation

x : Data set instance

$c(x)$: Class of x

C : Composite classifier

C_i : Component classifiers

$C_i(x)$: Prediction of C_i on input x

n : Number of component classifiers

M : Combining classifier

```
procedure train-stacked (C, T0)
loop for x ∈ T0
  loop for i from 1 to n
    train Ci on training set T0 − {x}
  collect [C1(x), C2(x), ..., Cn(x), c(x)] into T1
train (M, T1)
loop for i from 1 to n
  train (Ci, T0)

procedure apply-stacked (C, x)
x ← [C1(x), C2(x), ..., Cn(x)]
M(x)
```

Figure 2.2: Pseudocode for the stacked generalization algorithm.

that it received as input. This stacked generalizer showed an increase of approximately 20 percentage points in generalization accuracy over the best of the three level-0 classifiers. The accuracies of the three level-0 classifiers were 23%, 69% and 25%; the accuracy of the stacked system was 88%. Sejnowski and Rosenberg [1987] previously applied a neural network trained with backpropagation to this task, achieving 90% accuracy [2] .

Schaffer has investigated an extension of stacked generalization, *bi-level stacking*, in which the combining classifier also has access to the level-0 instance features as input, and not just access to the predictions of the combining classifiers [1994]. The intended benefit is to allow

---

[2] Wolpert observes [1992, p.253], "The purpose of this text-to-phoneme experiment wasn't to beat the performance ... of a metric-based HERBIE having access to all 7 input letters, nor even to beat the performance of back-propagation (i.e., NetTalk) on this data. Rather it was to test stacked generalization, and in particular to test whether stacked generalization can be used to combine separate pieces of incomplete input information.".

the combining classifier to take the original values into account to determine how to arbitrate among the component classifiers.

For example, assuming that the input representation includes the market trend, component classifier C_1 might be more reliable when the stock market is trending upwards, while classifier C_2 might be more reliable when the market is trending lower. In that situation, it might be advantageous to have the *trend* feature available to the level-1 classifier.

Schaffer compared stacking with bi-level stacking on five U.C.I. data sets [Murphy and Aha, 1994]. The three classifiers combined were a decision tree, rule induction and a neural network trained with backpropagation. The combining classifier was drawn from each one of these classes as well. In only three of fifteen experiments (using three combining classifiers on each of five data sets) did bi-level stacking outperform stacking. In a majority of the experiments (nine of fifteen), bi-level stacking was less accurate than stacking [Schaffer, 1994, p.57]. In three experiments, bi-level stacking and stacking yielded equal accuracy. This empirical result raises doubt about the utility of this implementation of bi-level stacking for most of the problems considered by Schaffer in these experiments. We speculate that including the original input vector may not be effective because the predictions of the component classifiers already adequately indicate that the instance falls in a specific local region of the instance space, and the inclusion of the original features does not give much additional information to the combiner. Partially in view of Schaffer's negative results, we do not explore bi-level stacking in this thesis.

2.2.1.2 Error-Correction Output Coding

Error-correction output coding (*ECOC*) may be seen as a variant of stacked generalization [Kong and Dietterich, 1995; Dietterich and Bakiri, 1995]. Rather than training each component classifier on essentially the same learning problem, ECOC trains a large number of components on a set of distinct problems and then combines their predictions. The algorithm works by transforming a multi-class problem into a large number of two-class problems. Each class is assigned a binary code word and each component classifier is assigned the task of learning one bit-position of that code word. The predictions of the components are collected into a vector, and a nearest neighbor algorithm is used to find the code word that is closest to the vector of predictions. A Hamming distance metric is used to compute the closest codebook vector to the vector of predictions. The class to which that codeword is assigned is taken as the prediction of the ECOC classifier. As will be seen in Chapter 5, this combining technique is similar to a stacked generalizer's combination of predictions using a 1-nearest neighbor algorithm, as suggested in a proposal for this research [Skalak, 1995]. There are notable differences between ECOC and the stacked generalization combining methods, however. In ECOC, the classes are assigned to code words *a priori*, and the code words can require "manual intervention" to create [Dietterich and Bakiri, 1995, p.272]. In stacked generalization, the mappings from classes to vectors of predictions are *learned* in the training phase, and no manual intervention is necessary to create code words.

The distinguishing characteristic of ECOC is that it treats the component classifiers as providing a distributed output representation for a concept. In particular, ECOC utilizes error-correcting codes for this representation. The benefit of using error-correcting codes is that a number of component classifiers may make errors in their output bits without causing the composite classifier to err.

Clear progress has been made by Dietterich and colleagues on utilizing error-correcting output codes for multi-class problems. However, the first implementations may suffer from several shortcomings, some of which are addressed by our dissertation work. On its face, the technique is applicable to multi-class problems, although we speculate that a two-class problem may be easily turned into a multi-class problem through the artifice of some (easy-to-learn) linear separation of one of the classes.

More to the point, our work responds to the call of Kong and Dietterich for level-1 representations that are "smaller and more efficient to evaluate" [1995, p.321] than the error-correcting codes that use hundreds of bits for some multi-class data. Level-1 code representations of that length require that hundreds of learning problems be solved. Examples of training 159 and 207 component classifiers are given by the authors.

In general, the outputs of a set of level-0 component classifiers do not demonstrate robust error correction in the manner guaranteed by error-correcting codes. The question arises whether error-correction is always needed, however. Part of the expense of ECOC arises from learning every pair of boundaries in a multi-class problem a large number of times. (Apparently, each boundary is learned a number of times equal to half the code-length [Kong and Dietterich, 1995, p.317].) Many boundaries for commonly applied data sets are unambiguous, however, and do not require many attempts at learning. Our Deliberate Misclassification algorithm, for example, examines inter-class boundaries by analyzing classifier confusion tables and determines which boundaries give rise to errors (Section 4.3).

The crucial similarities and differences between our own research and ECOC, as an important and useful classification method, should become more clear as this dissertation unfolds. Some of these points of comparison can be anticipated and summarized, however. The similarities include:

1. Both rely on classifier combination to solve classification problems.

2. Both can use a nearest neighbor algorithm to combine the predictions of a set of component classifiers. (Although in this dissertation, we usually rely on ID3 as a combining algorithm.)

3. With respect to the Deliberate Misclassification algorithm in Chapter 4, these two algorithms combine the predictions on a set of *distinct* sub-problems. (Entirely different methods are used to create the subproblems, however.)

.

The difference include:

1. In the applications described so far [Dietterich and Bakiri, 1995; Kong and Dietterich, 1995], ECOC often requires hundreds of component classifiers, which yield a level-1 representation that is not concise. We explore concise level-1 representations that are created from the predictions of only two or three component classifiers.

2. ECOC typically creates component classifiers that are more complex than obtained by training a single, standalone classifier on a task [Dietterich and Bakiri, 1995]. We investigate the combination of nearest neighbor classifiers that are less complex than a single, standalone nearest neighbor classifier trained on the same task.

3. ECOC is described as a method to solve multi-class problems, whereas our research is immediately applicable to both multi-class and two-class problems.

4. ECOC may require manual post-processing to create codes with more than 11 bits. Our combination techniques are automatic.

5. ECOC is not a classifier boosting algorithm, which is a major focus of this research (see Section 2.2.2).

.

2.2.1.3 Network Construction Algorithms

Several methods have been developed for dynamically adding to the architecture of a neural network. The analogy to the research described in this thesis is that whereas the neural net construction algorithms add nodes to a network (according to some criterion), our algorithm may add small nearest neighbor classifiers to a composite classifier (according to some criterion). Adaptive network construction algorithms are designed to create neural architectures that effect a balance between creating too many hidden units and too few hidden units. Too many hidden units can lead to overfitting; too few, to over-generalization. Connectionist algorithms that add units and connections to an architecture dynamically include the tiling algorithm [Mezard and Nadal, 1989], the upstart algorithm [Frean, 1990], cascade-correlation [Fahlman and Lebiere, 1990], and algorithms due to Marchand and colleagues [1990] and to Ash [1989]. The algorithms differ according to the configuration of the constructed network, the order in which units are added, and the criteria for unit addition.

The tiling algorithm [Mezard and Nadal, 1989] for network construction relies upon the same principle as the Inconsistency Reduction algorithm we have proposed for component classifier configuration [Skalak, 1995]. The principle is that in any layered architecture, the internal representation at a layer should be a *faithful* representation of the previous layer, and, hence, of the original data. A faithful representation is one in which two instances that are from distinct classes have a different value on at least one feature [Hertz *et al.*, 1991]. (Note that a faithful representation is not necessarily an identical one.) A representation that is not faithful is termed *inconsistent*. Inconsistent representations have different class assignments for at least one feature vector. In the tiling algorithm, a new unit is added to any layer that has an inconsistent representation. The new unit is trained only on the subset of patterns that given rise to the ambiguity. This process is iterated until the representation at each level is no longer inconsistent.

2.2.2 Boosting

The second composite classifier architecture we shall investigate is *boosting*. The goal of boosting is to increase the accuracy of a given algorithm on a given distribution of training instances. In computational learning theory, the given classifier is taken to be a *weak learner*, a learning algorithm that performs just slightly better than random guessing [Schapire, 1990]. Boosting is also related to stacked generalization, in that the given classifier is incorporated as a level-0 component in a stacked generalizer, and the goal is to create other components whose predictions will be combined with those of the given classifier (Figure 2.3).

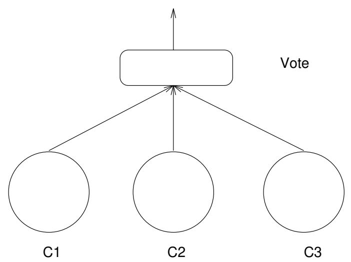

Figure 2.3: Boosting architecture. Classifier C1 is taken as given, and C2 and C3 are additional component classifiers. Their predictions are combined by majority vote.

Boosting is due to Schapire [1990] and has been investigated further by other researchers in computational learning theory [Kearns, 1988; Drucker *et al.*, 1994a; Freund and Schapire, 1995; Drucker and Cortes, 1996; Quinlan, 1996]. Schapire's boosting algorithm is a technique for creating a composite classifier, starting with a given component classifier, which we will call the *base classifier*, C_1. Boosting is a randomized algorithm that filters the training set to train two additional component classifiers, C_2 and C_3.

The training procedure is straightforward. Classifier C_1 is trained on a set of training examples. A fair coin is then tossed. If the coin comes up heads, then apply C_1 to new instances until C_1 *makes a mistake*. If the coin comes up tails, then apply C_1 to instances until C_1 *is correct*. In either case, add the triggering (mistaken or correct) instance to a training set for C_2. This process of flipping the coin and adding a training instance is repeated until sufficiently many examples have been selected to train C_2. Finally, new instances are drawn from a distribution of training instances, and are classified by C_1 and C_2. If they disagree, the instance is added to a training set for C_3. When C_3 has been trained on sufficiently many examples, the training stops.

To apply the composite classifier, a majority vote of C_1, C_2 and C_3 is taken. Boosting is defined within the *probably approximately correct learning* (*PAC-learning*) setting, in which such phrases as "sufficiently many examples" and "enough examples" are defined rigorously within the PAC framework. In PAC-learning, with high probability, a learning algorithm must output a correct concept hypothesis for all instances, except for a fraction of them that can be made arbitrarily small.

As it is typically applied, boosting is distinguished from stacked generalization in several ways. (1) Boosting assumes that a single classifier has been given, whose accuracy is to be increased. (2) The component classifiers are trained serially in boosting, so that results from the training of each classifier are used to form the training set for the succeeding component classifiers. (3) Voting is the usual means of combining categorical predictions in boosting, whereas stacked generalization often uses other combining algorithms.

Boosting thus yields a stacked generalizer that consists of three component classifiers, one of which is trained to arbitrate disagreements between the other two, and where predictions are combined by voting. Chan and Stolfo [1993a; 1995] have characterized this type of stacked generalizer created by the boosting algorithm as an *arbiter* architecture. The architecture has been studied in the context of learning from subsets of a large data set that is distributed across nodes of a network. A recent extension to boosting, called Adaboost (Adaptive Boosting) [Freund and Schapire, 1995], differs in that it can apply a larger number of weak learners, uses a weighted average to combine them, and changes the distribution of training examples as each weak learner is trained serially. In work published contemporaneously with our own, Freund and Schapire have attempted to use the new boosting algorithm, Adaboost, to improve a nearest neighbor classifier on a handwriting recognition task [1996]. The goal was not to increase accuracy, but to increase efficiency through reducing the set of stored prototypes. They demonstrated that a data set of 9709 training instances could be reduced to 2670 through the combination of 30 nearest neighbor classifiers. The error for the boosting algorithm was 2.7% versus 2.3% for a nearest neighbor classifier that used all instances as prototypes.

Another group of well-known researchers has encountered difficulties in boosting nearest neighbor classifiers. Drucker, Cortes, Jackel, LeCun and Vapnik [1994b, p.61] observed

> We have tried boosting on ... k-nearest neighbor classifiers. At the present time, we cannot explain why boosting does not work on these non-neural based classifiers.

Thus our efforts to boost nearest neighbor classifiers assume greater importance in view of this previously reported failure.

2.2.3 Recursive Partitioning

The third class of composite classifier architectures is recursive partitioning, also called divide-and-conquer. Recursive partitioning algorithms use a divide-and-conquer strategy to partition an instance space into regions that contain instances of only one class. A tree of component classifiers is created by recursively calling a construction function `form-classifier` on the instances that fall to each node [Brodley, 1994]. The `form-classifier` function determines if the instances are all from one class. If so, the class is returned. Otherwise, a classifier is trained on those instances and then applied to them to partition the instances for the next recursive step.

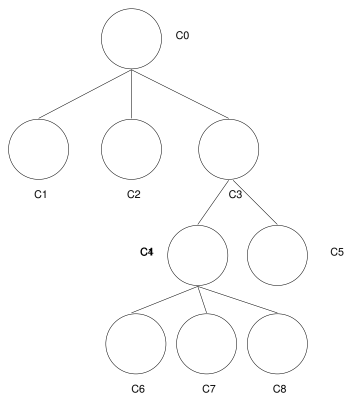

Figure 2.4: Hypothetical example of a recursive partitioning architecture. C0 to C8 are component classifiers, each of which applies only to a particular region of the instance space.

Recursive partitioning algorithms can be distinguished from *global scope* systems in which each component classifier makes a classification prediction for every instance and all the predictions are combined in some fashion. Although apparently no term has been coined for this class of algorithms, we will refer to them as *global scope* classifiers, because each component classifier is applied to all the instances in a data set. Stacked generalization and boosting yield global scope classifiers.

Work by Utgoff and Brodley provide paradigm examples of recursive partitioning combination algorithms. [Utgoff, 1989; Brodley, 1992]. Utgoff's *perceptron tree* algorithm combined a univariate decision tree with linear threshold units. The algorithm first determines if a subspace is linearly separable by using a heuristic measure. A linear threshold unit is applied if the subspace is linearly separable; if it is not, the space is split using an information-theoretic measure. Brodley's Model Class Selection system creates a recursive, tree-structured hybrid classifier, which combines decision trees, linear discriminant functions and instance-based classifiers. Brodley uses heuristic knowledge about the biases of machine learning algorithms to guide a search for classifiers that can solve subproblems of a classification problem with classifiers that match the bias of the algorithm to the subproblem.

The important advantage of this approach is that a classifier can be selected whose bias is appropriate for a region of the instance space. The result is an increased flexibility to draw on the special strengths of classifiers and of the concept description languages [Utgoff, 1989]. A second advantage is efficiency in application: not all component classifiers need to be applied to each instance, only those along a path through the tree need be invoked.

There are also disadvantages to the recursive partitioning. As a region is sub-divided, there are fewer training instances to fit. Fewer training instances to fit can result in overfitting of those instances and poor generalization accuracy. Overfitting is evident where the training set accuracy exceeds the test set accuracy by a large degree. From the standpoint of classifier combination, another potential disadvantage is that since only one classifier is ultimately used to classify each subspace, the possibility is foreclosed that other classifiers may provide useful predictions for instances in that subspace.

2.3 Component Classifier Construction

The last two sections dealt with the design considerations for composite classifiers and three architectures for combining classifiers. In this section, we descend from the architectures for combination to the component classifiers whose predictions will be combined. We organize previous research in this area according to a taxonomy of methods that have been used to create diverse components. Recall that at least to some degree, component classifiers must be diverse — have to make some different errors — to be combined effectively. Of course, as with all these methods, the generated component classifiers must not only make diverse predictions, but must make predictions in such a way that the combining algorithm can learn from them.

While there has been a fairly large amount of work on classifier combination, the techniques used to create the components fall into a fairly small number of clusters:

1. Reasoning strategy combination
2. Divide and conquer
3. Model class combination
4. Architecture and parameter modification
5. Randomized search
6. Feature selection
7. Training set resampling

We shall review the work that falls into each of the above categories. The section on diverse reasoning strategies may be regarded as general background: selecting diverse classifiers is a narrow version of the wider question of how to combine diverse modes of reasoning (e.g., combining rule-based and case-based reasoning [Rissland and Skalak, 1991], multi-strategy learning [Michalski, 1994].) Two sections deal with techniques that have myriad applications in artificial intelligence and in computer science in general: divide-and-conquer and search. Both these general approaches have been applied to classifier selection. A more task-specific technique for creating diverse classifiers has been parameter modification. Since almost all classifiers depend on one or more parameters, one way to create diverse classifiers is to choose the parameters differently. Feature selection is often a crucial aspect of inductive learning, and selecting classifiers that apply different features also is a way to create classifiers that make different predictions. Finally, Breiman has advocated training set resampling to achieve diversity. Our review emphasizes any work done using instance-based algorithms.

2.3.1 Reasoning Strategy Combination

The combination of case-based systems and other reasoning systems has been an active area of research. Hybrid case-based systems have been built in a variety of application domains to perform a number of tasks (e.g., CABARET (case-based module combined with rule-based module; the shell is instantiated in a legal domain) [Rissland and Skalak, 1991], KRITIK (case-based and model-based modules combined for mechanical design) [Goel, 1989], FRANK (case-based and OPS5 system in planning framework for diagnosis) [Rissland *et al.*, 1993], IKBALS-II (case-based and rule-based system for law) [Vossos *et al.*, 1991], and MEDIATOR (case-based and decision-theoretic modules combined for labor negotiation) [Sycara, 1987]). Case-based systems that have combined reasoning strategies for classification include ANAPRON (case-based and rule-based system for word pronunciation) [Golding and Rosenbloom, 1991], CASEY (case-based and model-based reasoning combined for heart disease diagnosis) [Koton, 1988] and PROLEXS (case-based and rule-based system for Dutch landlord-tenant law) [Walker, 1992]. These systems do not, however, combine multiple case-based classifiers, which would be the appropriate analogy to our proposed research.

The emphasis in these projects was frequently on the control strategy or the implementation framework for integrating the component reasoning modules. In most of these efforts, the control regimes were more sophisticated than the straightforward control strategies contemplated by the composite classifiers we study. Hybrid case-based systems require an effective control strategy because they typically apply larger amounts of domain knowledge, incorporate more extensive case representations, or perform more complex tasks than many learning systems. Typically, a case-based reasoning system also incorporates a measure of case similarity more suited to a particular application than the generic, default Manhattan or Euclidean metrics applied by many nearest neighbor classifiers.

2.3.2 Divide and Conquer

One way to achieve classifier diversity is to make them applicable to separate regions of the space of instances. As we have noted, this strategy is called divide-and-conquer or recursive partitioning. It is a general strategy for creating disjoint regions of classifier applicability. As we discussed in Section 2.2, Utgoff's perceptron tree algorithm and Brodley's MCS system use this approach to find regions of the instance space in which their component algorithms are selectively superior [Utgoff, 1989; Brodley, 1994].

The *hierarchical mixtures of experts (HME)* algorithm of Jordan, Jacobs and Barto [Jordan and Jacobs, 1993; Jacobs *et al.*, 1991] also uses divide-and-conquer to create diverse component classifiers. In the HME algorithm, expert component networks compete to learn training patterns and a gating network learns to mediate this competition. The gating network learns a soft partition of the instance space at the same time that the expert networks are trained. The hallmark of this approach is that the HME algorithm enables *task decomposition* — the combination algorithm learns which expert network to devote to which input. Nowlan has proposed a similar approach to dividing a task among a set of component neural networks [1990]. Radial basis functions also effect a soft partition of the input space [Ripley, 1996]. Unlike the

HME algorithm, which trains in a single phase, radial basis function networks are typically trained in two phases [Ripley, 1996, p.131]. Radial basis function centers are located in a first training phase. In a second phase, the coefficients of the basis functions are determined. The coefficients determine how to combine the basis functions to predict output values.

Cooper and his colleagues have been granted a patent on a classification algorithm called *NLS*, an acronym for *Nestor Learning System* [Cooper *et al.*, 1982; Wasserman, 1993]. NLS consists of an array of so-called Restricted Coulomb Energy System (RCE) classifiers whose classification decisions are combined using a proprietary Class Selection Device [3]. An RCE classifier is a descendant of classifiers that are inspired by electrostatic field theory. Like a nearest neighbor classifier, it is an explicitly local classifier. To create an RCE classifier, a training procedure sequentially fits *basins of attraction* around a subset of training instances. In the classification phase, if a test instance falls within the basin of attraction of a training instance, it is given the class of that training instance. During classification, the highest priority RCE classifier that gives an unambiguous response is the prediction of the composite algorithm.

Chan and Stolfo [1993b] also report results from combining a set of classifiers in a binary tree structure, where the classifiers are a memory-based classifier called WPEBLS, based on the PEBLS memory-based classifier [Cost and Salzberg, 1993]. PEBLS is a nearest neighbor classifier that uses a minor variation of the Stanfill-Waltz metric for symbolic attribute values [Stanfill and Waltz, 1986]. Chan and Stolfo assume a problem framework in which data are distributed across nodes of a network of workstations. Their goal is to show that learning can be done in parallel using proper subsets of the data for training. Chan and Stolfo compare several methods for creating a distribution of training instances that are used to train "arbiter" classifiers. The role of arbiter classifiers is to combine the predictions of other component classifiers. On two tasks, Chan and Stolfo demonstrate that combining the predictions of distributed classifiers trained on partitions of the data often yields accuracy almost as high or as high as when all the data are present at one network node. This demonstration shows that

[3] Nestor Learning System and Restricted Coulomb Energy classifiers are trademarks of Nestor Inc.

data sets that are too large to store at a single node may partitioned across network nodes. Component classifiers may be trained at local nodes, on subsets of the data, and the predictions of these classifiers may be combined to yield an accurate composite classifier.

Edelman creates a stacked generalizer where the predictions of a set of level-0 radial basis function classifiers are combined using a level-1 radial basis function classifier [1995]. (Radial basis function classifiers are described briefly in Section 2.5.1.3.) This composite architecture is applied to the task of face recognition; each network is trained to recognize a single prototype, a specific face. Edelman shows experimentally that the dimension of a category may be characterized by the number of stored prototypes, rather than by the dimension of a raw or abstract feature space.

2.3.3 Model Class Combination

Another approach to building a diverse set of classifiers is to use classifiers from different model classes, such as decision trees, nearest neighbor algorithms and linear discriminant functions. For example, multilayer perceptrons and learning vector quantization have been combined [Battiti and Colla, 1994]; so have a statistical classifier, memory-based reasoner and a neural network trained with backpropagation [Zhang *et al.*, 1992]. It is an open question whether classifiers from similar or dissimilar model classes are combined most effectively, and how to measure similarity for this purpose. The answer will depend a number of factors, including the extent to which classifiers drawn from the model classes make independent errors and on the level-1 combining algorithm.

In a set of experiments that apply a number of different techniques for creating diverse classifiers, Breiman has investigated stacked generalization for function approximation in an approach he calls *stacked regression* [1992]. Breiman also investigated the combination of several types of predictors: Classification and Regression Trees (CART trees) and two types of linear regression: subset regression and ridge regression. Diverse CART trees were produced by growing a large tree and then pruning it to the k-node subtrees that had the minimal error on the training set, for various choices of k. The values of k were not reported, but "about 50" subtrees were combined in experiments on two data sets [Breiman, 1992, p.8]. Reductions in error were reported on both data sets over the single tree with the lowest error rate determined by cross-validation.

Subset regression uses subsets of features in a linear regression function. In the subset regression experiments, Breiman combines predictors constructed by stepwise backward elimination of variables. In stepwise backward elimination, all the possible variables are included in the initial model; they are then eliminated one at a time [Krzanowski, 1988]. A statistical test is used to eliminate variables that do not make a significant contribution to the reduction in the error of the model.

Ridge regression is a linear-regression variant that is used for highly correlated variables, as is often the case for a set of predictors that are designed to approximate the same function [Ripley, 1996; Breiman, 1992]. Ridge regression adds a constraint that the sum of the squares of the regression coefficients be equal to a constant λ. Varying this parameter produces a set of predictors. Breiman notes that he has also stacked linear regression predictors with k-nearest neighbor predictors, and reports "substantial reductions in error." A further description of these results will not be forthcoming, however[4].

2.3.4 Architecture and Parameter Modification

As in Breiman's experiments with ridge regression, where an algorithm depends on one or more parameters, different classifiers can be created by varying those parameters. For instance, neural networks typically require that a large set of parameters be initialized, for example, the learning rate and the network weights. Perrone has combined neural networks that are different only by virtue of having different random weight initializations [1993]. Maclin and Shavlik have shown that competitive learning can be used to initialize neural network weights to increase the number of local minima that may be reached by the networks, thereby increasing their potential diversity [1995]. (Competitive learning is an incremental technique that is used

[4] Personal communication.

to find a given number of "average" instances. Beginning with a set of random instance seeds, simple competitive learning moves a seed closer to a training instance each time a seed is the closest one [Hertz *et al.*, 1991; Maclin and Shavlik, 1995].)

The number of (input, output and hidden) nodes are parameters for many neural network algorithms. A similar issue arises in pruning a decision tree. Determining the number of nodes in a decision tree or the number of computational units in a neural network may be more a matter of architecture selection than of parameter setting, but the point is the same. Different classifiers will result from setting the design parameters differently.

Number of Neural Units. Investigations of the effect of the combination of networks with different numbers of units have been performed by Battiti and Colla [1994] and Perrone [1993]. Battiti and Colla vary both the architectures of the multilayer perceptron networks (the number of input and hidden nodes) and the random weight initializations in an attempt to create networks with uncorrelated errors. They observe that the predictions of the resulting networks are not independent, although their theoretical analysis makes the common assumption of independence. On a handwritten character recognition task, Perrone observed decreases in mean-squared error for both the best individual network and an ensemble of networks as the number of units was increased to 40.

The usefulness of randomly varying weight initializations and network topologies has been questioned by Opitz and Shavlik [1995], who found that bagging, described in Section 2.3.6, outperformed such random methods for classifier construction.

Number of Decision Tree Nodes. Decision-tree pruning has been used to generate classifiers that make diverse predictions. Breiman [1992] combines a set of trees that have been pruned to the k-node trees that displayed the smallest training set error, for various choices of k. Breiman showed that stacking trees outperformed choosing the tree with the best cross-validation performance on two datasets. Oliver and Hand [1995] suggest that the instance to be classified should determine in part which component tree predictions should be combined.

2.3.5 *Randomized Search*

Randomization can also be exploited to create diverse classifiers. In an extensive series of experiments, Ali and Pazzani [1995; 1996] have evaluated the combination of stochastically generated decision trees. At each decision tree node, a test was randomly selected from among those whose information gain is within a fixed interval of the gain for the optimal split. Applied to 29 data sets from 21 domains, Ali and Pazzani showed significant reductions in error by combining 11 stochastically generated decision trees through voting [Ali and Pazzani, 1995]. Geman, Amit and Wilder later proposed the idea as well [Geman *et al.*, 1996].

Opitz and Shavlik use a genetic algorithm to create accurate and diverse neural networks to be used as components [1995]. As with other genetic algorithm applications, their ADDEMUP algorithm applies mutation and crossover operators to a population of neural networks. The operators perturb and combine the topologies of the member networks. The hallmark of ADDEMUP's innovative approach is a fitness function that maximizes an adaptive, but linear, combination of accuracy and diversity terms.

In a similar scheme, an evolutionary algorithm has been developed by English and colleagues to evolve a collection of level-0 recurrent networks to be applied in a stacked generalizer for time-series prediction [English and Gotesman, 1995; English, 1996]. Diverse models are created by mutating network topologies and parameters, as well as by mutating hyper-parameters that govern the precision of the parameters used to specify the network.

2.3.6 *Training Set Resampling*

Training component classifiers on different samples of the training set provides another means to achieve diversity. The foremost example of this technique is Breiman's *bagging* algorithm [1994a]. Bagging, which is an acronym for "bootstrap aggregating," is a form of stacked generalization in which bootstrap sampling — random sampling with replacement — is used to create different training sets for a set of predictors. Once the classifiers are trained, their predictions are aggregated by averaging numeric predictions or voting symbolic ones.

In Section 2.2.2, we noted that a group of well-known researchers reported that they were unable to boost k-nearest neighbor classifiers [Drucker *et al.*, 1994a]. In a second reportedly unsuccessful experiment to create a composite classifier with nearest neighbor components, Breiman [1992] observed that bagging 100 nearest neighbor classifiers did not reduce the rate of misclassification below that of a single nearest neighbor classifier. He attributed this failure to a characteristic of nearest neighbor classifiers: that the addition or removal of a small number of training instances does not change nearest neighbor classification boundaries very much. Decision trees and neural networks, he argued, are in this sense less *stable* than nearest neighbor classifiers.

In this thesis we use nearest neighbor classifiers that store a very small number of prototypes in part to make classifiers *unstable*, so that they are *more* suited for combination.

Boosting also relies on training set sampling to create diverse classifiers [Schapire, 1990]. In Schapire's boosting algorithm (Section 2.2.2), the original training set is stochastically filtered to train an additional classifier, according to a coin toss and to whether the original classifier makes an error. The approach of Chan and Stolfo for combining predictions from distributed data [1993b] also requires training set resampling and filtering to train arbiter classifiers.

2.3.7 Feature Selection

Even if classifiers are from the same model class, if they use different input features, they are potentially diverse. Feature selection generally is an important aspect of representation, and representation is a crucial aspect of learning. Therefore it is probably not surprising that feature selection may be applied to create ensembles of classifiers as well as to create an independent classifier.

The few projects that have used feature selection to achieve diversity have already been described for other reasons. In their study of voting and related combination algorithms for combining the predictions of neural networks trained to recognize handwritten characters, Battiti and Colla observed that qualitatively different input features lead to the largest increases in accuracy [1994]. Breiman successfully combined predictors created by stepwise backward elimination of variables [Breiman, 1992] (Section 2.3.3).

Lastly, in Wolpert's NetTalk experiment, the components were distinguished only because they were each given a different input feature from the representation of training instances [1992]. Recall that the NetTalk task is to learn the mapping from a window of letters to a phoneme for the letter in the center of the window. The stacked generalizer incorporated three nearest neighbor components, each of which was given (by hand) a one-dimensional input space. Out of each seven-letter window, one classifier received as input only the third letter; the other classifiers received the fourth and fifth letters as input. The output was a prediction of the phoneme corresponding to the fourth letter, the middle of the window. The classifiers that used the third and the fifth letters as input to predict the phoneme for the fourth letter were poor, but still managed to increase the accuracy of the classifier that received the fourth letter as input (Section 2.2.1.1).

2.3.8 Discussion

We have identified seven classes of methods for creating component classifiers that are diverse. Implementations of these methods have attained varying degrees of success, and, at this point, no extensive comparisons have demonstrated that any method is more likely to yield effective component classifiers. There are many ways that research progress might be made from this point, but three paths are clear: (1) explore in depth a class of methods for creating diverse components; (2) amalgamate previously identified methods; or (3) create a new class of methods.

In this dissertation we attempt to amalgamate existing methods of inducing diversity, but we also suggest and explore a new class of methods. As for amalgamating existing approaches, there is much more work that might be done. Since our research relies on nearest neighbor classifiers that store a few prototypes, it falls under the heading of architecture modification. We also rely on a form of classifier sampling to create nearest neighbor classifiers that use a small number of prototypes. This sampling represents a type of random search method for classifier creation; sampling is perhaps the ultimate random search technique. We show that sampling classifiers, through random selective sampling of prototypes, can yield classifiers of high accuracy. Finally, nearest neighbor classifiers that store only a few prototypes may be seen as a severe form of training set resampling, as in the *Coarse Reclassification* algorithm that we shall describe in Section 4.2.

Our work on combination also has yielded a new approach to creating diverse components. All of the approaches that we have described modify classifiers or resample the training set to achieve diversity. None of these approaches takes account of a fundamental aspect of our task: supervised learning. Each training instance has a class label associated with it. We have control over this class label and may vary the class assignments to train the diverse classifiers. Training classifiers to learn different, but similar, concepts by changing certain class labels, appears to be a useful — and untried — way to achieve diversity. This hypothesis is behind the *Deliberate Misclassification* algorithm that we shall describe in Section 4.3.

2.4 Combining Classifier Construction

Having reviewed seven general methods for creating component classifiers, we move to the level of the combining classifier. We survey approaches that have been taken to combining the predictions of the component classifiers. The section is structured according to whether the components output a real value (as for function approximation) or a class label (as for classification). Since this thesis does not deal with general function approximation, we examine that literature in less detail. This section is structured according to the following outline:

1. Function approximation
   (a) linear
   (b) non-linear
2. Classification
   (a) voting
   (b) non-voting

To anticipate our survey, the overwhelming amount of research in function approximation has relied on a linear combination of classifiers. The bulk of the research on categorical classification has used voting as a combination method. The reliance on these two techniques may be surprising in view of their shortcomings.

2.4.1 Function Approximation

2.4.1.1 Linear

Where classifiers output a real-value prediction, a linear combination function is a common combiner [Breiman, 1992; Hashem, 1993; Oliver and Hand, 1995; Maclin and Shavlik, 1995; Tumer and Ghosh, 1996]. The linear combination may be simple, unweighted averaging or a weighted combination of predictions. Simple linear regression algorithms are often used to compute weights on the component classifier predictions.

One advantage of linear combination methods is that it does make formal analyses easier. Perrone has shown that under certain conditions on an error function, an optimal set of coefficients for a linear combination of a set of predictors can be derived theoretically [1993]. In experiments on an optical character recognition database, the attempt to compute these coefficients proved impossible due to difficulties in computing the inverse of a covariance matrix of the predictor estimates. In practice, the large assumption was made that the errors made by the different predictors were uncorrelated.

A second advantage of linear methods is that they often seem to work well. In his survey of 200 papers combining economic and meteorological forecasts, Clemen has observed that "Trying ever more elaborate combining models seems only to add insult [to] injury, as the more complicated combinations do not generally perform all that well." [Clemen, 1989, p.566].

A third advantage of simple averaging is that it can lower the variance of the composite prediction. Mani has observed that this result follows from an analogy to stock portfolio diversification [1991], and argues for creating a portfolio of neural networks.

On the other hand, a simple linear combination cannot reflect the local expertise of a component classifier for an instance in a particular region of the feature space. Each component's contribution is the same, regardless of the input. Linear functions fail to support local expertise, an important aspect of classifier combination.

Also, since the predictions of a set of component classifiers are likely to be highly correlated, Breiman states that a regression technique should be applied that has been designed to work with highly correlated variables, such as ridge regression [1992], and ridge regression is rarely used as a level-1 combining algorithm. Hashem argues that using an ordinary least squares linear combiner can lead to regression coefficients that are themselves highly correlated, which leads to problems in the interpretation of the effect of each component classifier upon the composite output as well [1993].

2.4.1.2 Non-linear

Despite the work on combining non-linear *component* classifiers, non-linear combining algorithms have received much less attention [Zhang *et al.*, 1992; Edelman, 1995; Jacobs *et al.*, 1991; Jordan and Jacobs, 1993]. Non-linear combining functions are usually implemented as neural networks, such as multilayer perceptrons or radial basis function networks. There is no *a priori* reason to believe that linear methods are adequate to model the combination of arbitrary function approximators, and there is evidence to the contrary. Due to the *selective superiority* [Brodley, 1994] of classifiers in different regions of the instance space, it is expected that the components will have different relative contributions in different parts of the space. A linear function of the classifier predictions alone cannot track the selective contributions of the components.

2.4.2 Classification

2.4.2.1 Voting

Just as linear regression is the default method for combining function approximators, plurality voting is the default method for combining classifiers [Breiman, 1994a; Kwok and Carter, 1990; Hansen and Salamon, 1990]. Two of the most extensive analyses have been done by Ali and Pazzani and by Battiti and Colla.

Ali and Pazzani [1995] have experimented with the combination of stochastically generated rules and decision trees using voting and three probabilistic algorithms as combiners, called "Bayesian Combination", "Distribution Summation" and "Likelihood Combination".

The general form of Bayesian Combination uses Bayes Rule to assign to each test instance the class that maximizes the degree of belief that an instance belongs to that class. As implemented, the degree of belief for a class for a set of models is a sum (over the models) of the accuracy of each model times the posterior probability of the model (the probability of a model given a training set) [Buntine, 1990].

Distribution summation associates a vector of length s with each decision-tree leaf, where s is the number of classes, with one entry for each class. For each leaf in each tree, the vector records the number of training examples from each class that fall to the leaf. To classify a new instance by an ensemble, each component decision tree filters the instance to a leaf. The vectors corresponding to the leaf for each component tree are added component-wise. The prediction for the instance is the class with the value in this summed vector.

Likelihood combination is based on the "logical sufficiency" of each class and each component decision-tree leaf. Logical sufficiency is a ratio of (i) the probability that a random instance falls to a given leaf and is from a given class to (ii) the probability that a random instance falls to a given leaf but is not from the given class. Bayes Rule is used to combine the logical sufficiencies. The class assigned to a test instance is the class with the highest posterior odds for that class given the set of decision tree models. Details for all these combining methods are given in the cited reports. In an empirical test using twenty-nine datasets (some being variants of others), voting was the most accurate combiner, notwithstanding the relative sophistication of the other combination methods.

Battiti and Colla [1994] have analyzed several combination mechanisms for combining the classifications of multilayer perceptrons trained to recognize handwritten digits. The combination schemes compared were unanimity, majority vote, averaging raw output activation values and several mechanisms based on thresholding the confidence of the component networks in their classification predictions. Analysis and experiments were performed in the usual framework for character recognition, where a reject option is available, if the confidence (however measured) in the prediction is sufficiently low. (A reject option typically puts an example in an "unknown" bin, where a human classifies the example.) Unanimity rejected a pattern unless all the classifiers agreed on a prediction. Majority vote outperformed a combination rule requiring unanimity of prediction among the components.

Computational learning theory and voting. One advantage of voting is that it lends itself to theoretical analysis, and so voting has received attention from the computational learning theory (COLT) school of research. Angluin describes the goal of computational learning theory as "Give a rigorous, computationally detailed and plausible account of how learning can be done." [1992, p.351]. She specifies voting schemes as one class of techniques for constructing learning algorithms.

A typical theoretical setting for the COLT approach is that a finite pool of learning algorithms ($\mathcal{A}$) is given and a master algorithm must be designed to combine the predictions of the pool, where the master algorithm (sometimes called the "fuser" or the "fusing algorithm") has access to the predictions of each of the pool algorithms but not to the sample input. Assumed are a two-class problem and on-line learning, where a series of examples is put to the learning system, and a reinforcement is given after each example's presentation as to whether the prediction of the class of the example is right or wrong. The goal of each algorithm is to reduce the pool to a subset of reliable algorithms, or, more generally, to weight the algorithms in the pool.

Rao, Oblow, Glover and Liepins [Rao *et al.*, 1994] distinguish two distinct problems in trying to combine a set of PAC (probably approximately correct) learning algorithms [Valiant, 1984]: *open fusion*, where the fuser is given the training examples and the hypotheses of the individual learners, and *closed fusion*, where the fuser cannot access the training instance or the hypotheses of the individual classifiers. In the closed fusion case, assuming statistically independent learners, Rao and colleagues demonstrate analytically that a linear threshold function of the classification decisions of the individual learners can result in a system where the "confidence parameter of the entire system can be made greater than or equal to that of a best learner." (p. 326).

Open fusion is similar to bi-level stacking in that the combining algorithm has access to the level-0 training examples [Schaffer, 1994]. As we discussed in Section 2.2.1.1, Schaffer's experiments showed that bi-leveling stacking did not yield an improvement over standard stacking. Therefore, in this dissertation we employ the *closed fusion* approach.

The *Halving* and *Weighted Majority* algorithms of Littlestone and Warmuth [Littlestone, 1987; Littlestone and Warmuth, 1989] are the two seminal algorithms that use forms of voting as combination techniques. The Halving combination algorithm makes a prediction according to the majority of the consistent functions in the pool, where a function is *consistent* if it has correctly predicted all the examples in the sequence of instances previously put to it. Of course, as the sequence of examples gets longer, the number of consistent functions tends to decrease. If there is a consistent algorithm in the pool, the Halving algorithm finds it after making no more than $log_2|\mathcal{A}|$ mistakes, since at least half the algorithms are removed from the pool when a mistake is made.

The Weighted Majority (WM) algorithm is an extension of the Halving algorithm, where a weight is associated with each algorithm in the pool. WM predicts according to whether the total weight of the algorithms in the pool predicting 0 is greater than the sum of the weights of the algorithms predicting 1. The learning rule is simple. Each time WM errs, the weights of the erring algorithms in the pool are multiplied by a factor $\beta \in [0, 1)$, which has been fixed in advance. The WM algorithm reduces to the Halving algorithm for $\beta = 0$. Littlestone demonstrates a $O(log|\mathcal{A}| + m)$ upper bound on the number of mistakes made by the composite algorithm, if there is an algorithm in the pool that makes at most m mistakes.

Limitations of voting. Voting is relatively simple to analyze, and depending on the user (and the decision being made), seems a reasonable way to combine decisions when one has no reason to elevate the decision of one voter over another. So the default combining method for this problem and other categorical problems is the plurality vote.

Nevertheless, despite voting's appeal, the technique has several shortcomings. The first is that voting is unable to account for local expertise, by appealing to the prediction of a minority of classifiers, or of a single classifier that is more likely to be correct for a given instance.

Intuitively, voting the predictions of a large set of reasonably accurate classifiers would seem to produce a very accurate composite classifier. However, in general, increasing the set of components and voting their predictions does not reliably lead to higher generalization accuracy [Kearns, 1988; Schapire, 1990; Heath *et al.*, 1996]. In brief, the difference between a good classifier and a superior one is determined by their behaviors on difficult instances; both will probably get the easy ones right. If an example is difficult to classify, then the average classifier will get it wrong, and taking a majority vote of classifiers that are more likely to get it wrong will not increase, and may decrease, classification accuracy.

Voting requires more than two component classifiers. While this may seem like a trivial requirement, it does prevent the application of voting to the simplest, and most efficient, composite classifier architectures, which incorporate only two component classifiers. Several results have established the greatest incremental gain for the combination of only two components [Battiti and Colla, 1994; Perrone, 1993]. For some real applications, potentially the optimal tradeoff between accuracy and computational expense may be made with only two components (Section 2.1.3.1).

Paradoxes are also lurking in the background of voting. Some classifiers associate a probability with each class, and can therefore output a ranked list of class predictions, rather than a single prediction. Arrow has shown generally that where a collection of individuals ranks a set of preferences, there is no way to amalgamate those preferences in a way that satisfies a set of apparently reasonable axiomatic constraints, as shown by Arrow's General Possibility Theorem [Arrow, 1963]. In particular, there is no way to achieve a group ("social") ranking that reflects the individual rankings in a reasonable way. This result provides a fragile backdrop to attempts to combine the set of class rankings produced by a set of classifiers into a composite ranking. Although the result takes some time to derive formally, Arrow briefly restates his result informally:

> If we exclude the possibility of interpersonal comparisons of utility, then the only methods of passing from individual tastes to social preferences which will be satisfactory and which will be defined for a wide range of sets of individual orderings are either imposed or dictatorial.
>
> The word "satisfactory" in the above statement means that the social welfare function does not reflect individuals' desires negatively ... and that the resultant social tastes shall be represented by an ordering having the usual properties of rationality ascribed to individual orderings .... [1963, p. 59]

The terms *imposed* and *dictatorial* are two crucial terms in this restatement of the theorem. Arrow uses *imposed* to mean that "there is some pair of alternatives x and y such that the community can never express a preference for y over x no matter what the tastes of all the individuals are" [1963, p. 28]. Arrow uses *dictatorial* to mean that "whenever the dictator [an individual] prefers x to y, so does society" [1963, p. 30]. It has been observed that ranking cases by similarity to a new problem according to each feature and then amalgamating the rankings can run afoul of Arrow's General Possibility Theorem [Skalak, 1990].

2.4.2.2 Non-voting Methods

Implemented non-voting methods applied to classifier combination include ranking algorithms [Ho *et al.*, 1994], nearest neighbor [Wolpert, 1992; Dietterich and Bakiri, 1995], and an algorithm based on the Dempster-Schafer theory of evidence [Mandler and Schurmann, 1988; Xu *et al.*, 1992]. Character and handwriting recognition are the application domains in which less conventional approaches have been tried [Mandler and Schurmann, 1988; Ho *et al.*, 1994; Xu *et al.*, 1992; Battiti and Colla, 1994]. In these applications, which have been the object of much research in pattern recognition, a more flexible problem framework than "straight" classification is usually posed. Additional problem dimensions include ranking predictions, rather than giving a single, most likely, prediction, and the option to reject a pattern, which corresponds to giving an "unknown" answer.

Wolpert used a nearest neighbor combiner in a single NetTalk example, as previously discussed [Wolpert, 1992]. An l_1 metric is used by Dietterich and Bakiri in error correcting output coding to compute the distance between each static codeword and a vector of the outputs of the classifiers used to predict each bit in the codeword [Dietterich and Bakiri, 1995]. (Recall the discussion of error-correcting output codes in Section 2.2.1.2.)

Amalgamating rankings may be seen as a generalization of voting, and this extension has been investigated by Ho and colleagues for handwriting recognition [1994]. In particular, a consensus mechanism called the *Borda count* has been used to amalgamate the predictions of classifiers, each of which outputs a ranked list of predictions in order of the confidence of the classifier in each predicted class. As used by Ho, each class has a Borda count given by "the sum of the number of classes ranked below it by each classifier" [Ho *et al.*, 1994, p.68]. The class with the highest Borda count is the ultimate prediction.

Xu and colleagues make two suggestions for deriving probability estimates from a k-nearest neighbor classifier: (i) the ratio of the number of neighbors k_i from class S_i to the total number of neighbors k; and (ii) the reciprocal of the distance of the test case from each of the classes, normalized by the sum of those reciprocals [Xu *et al.*, 1992]. So if $d_k(i)$ represents the distance of test case x from each of the classes $S_i, i = 1, \ldots, s$ for a nearest neighbor classifier C_k, the probability of class S_i is

$$p_k(i) = \frac{1/d_k(i)}{\sum_{i=1}^{s} 1/d_k(i)}$$

Once probabilities are obtained, other probabilistic evidence combination techniques could be used. Their suggestions have not been investigated experimentally by the authors, however.

Rather than making a (scalar) probability estimate to a single proposition, the Dempster-Shafer theory of evidential reasoning assigns a probability interval to sets of propositions [Dempster, 1968; Shafer, 1976]. The interval reflects the strength of the evidence in favor of the set of propositions, but also reflects the amount of information that is available. In a classifier combination context, the sets of propositions correspond to statements that a component classifier C_i predicts class label S_j on input x. The theory then provides a mechanism to reason with such interval estimates to determine the strength of evidence for a class prediction. A disadvantage of the framework is that it can require a large amount of computation [Mandler and Schurmann, 1988]. Dempster-Shafer Theory has also been criticized as giving results that are counterintuitive [Ginsberg, 1993].

2.5 Nearest Neighbor Classifier Construction

In the last sections, we reviewed the methods for creating component and combining classifiers from a variety of model classes. In this section, we study algorithms that have been used to build the particular type of component classifier with which we are concerned: nearest neighbor classifiers. We concentrate on prototype selection. While there are other aspects of a k-nearest neighbor classifier that can be varied to achieve a design goal, such as the distance metric or the neighborhood size, some success in building accurate and efficient independent nearest neighbor classifiers has been shown by selecting appropriate prototypes.

Our review is somewhat lengthy due to the large number of prototype selection algorithms that have been suggested. All but a few of these algorithms follow a very similar procedure. They filter training instances according to a function that depends on two criteria: (i) whether the instance is being classified or is doing the classifying (by virtue of its proximity to a test instance), and (ii) whether the classification in (i) is correct or incorrect. Minor variations of these two dimensions generate many of the algorithms.

These algorithms are also of circumscribed interest because, for a given presentation of training data, they are suitable only for finding prototypes for one, independent nearest neighbor classifier. Our goal is to create algorithms suitable for locating prototypes for a set of component nearest neighbor classifiers. Nevertheless, these algorithms are an important part of the historical background for our research.

"Prototype" has a variety of tangled meanings. Different groups of researchers have adopted distinct meanings for this term. Whereas in cognitive science the term refers to an example that is ideal or distinguished for any one of a number of reasons, in pattern recognition the term refers to any reference instance stored by a nearest neighbor classifier. A nearest neighbor classifier computes the similarity with respect to these prototypes in order to predict the class of previously unseen instances.

We will call a nearest neighbor classifier that incorporates all training instances as prototypes a *full* nearest neighbor classifier. While we hesitate to use our own neologism, no standard term seems to have been adopted.

"Prototype" is ambiguous because the connection between the cognitive-science and pattern-recognition notions of a prototype has not been precisely characterized. To establish a connection, one could establish whether the ideal examples for a domain are the ones that are best retained as reference instances in a classifier, but this inquiry is outside our immediate interest in classifier combination.

In this section, our main goal is to review the methods that have been used to select prototypes for a single, independent nearest neighbor classifier. Before we consider that research, we give a short description of the functions that prototypes have served in related fields, including cognitive science, case-based reasoning, and the theory of local learning algorithms.

2.5.1 Prototype Roles

2.5.1.1 Cognitive Organization

In cognitive science, prototypes are the examples of a category that are best in some sense, such as the most typical. Minsky has observed, for example, "The problem of reading *printed* characters is a clear-cut instance of a situation in which the classification is based ultimately on a fixed set of 'prototypes'." [1965] [emphasis in original]. Prototypes have been advanced as the basis for cognitive theories of category representation, category learning, and classification. In particular, a theory of concepts based on prototypes is a response to the inadequacy of the classical model of categorization [Smith and Medin, 1981].

The classical model of categorization holds that a necessary and sufficient set of criteria determine whether an object is a member of a category. While this rule-governed model survived unchallenged for a long time, psychological studies and philosophical investigations have demonstrated that often some examples of a category are better than others, undercutting the classical theory [Rosch and Mervis, 1975; Smith and Medin, 1981].

Wittgenstein was one of the first philosophers to undermine the classical theory of categorization. In a famous example, Wittgenstein observed that there was apparently no set of necessary and sufficient features that in general characterize the abstract category *game* [Wittgenstein, 1953]. He coined the term *family resemblance* to characterize the relationship among various examples of games, a term later used by Rosch and colleagues [Rosch and Mervis, 1975]. Wittgenstein used the example of the concept *game* to demonstrate that categories can have extensible boundaries and that categories can have both central and non-central members.

One response to this demonstration was the *prototype model* of categorization. In the prototype model, sometimes called *prototype theory*, a new example is classified as in or out of a category by computing a weighted similarity of the features possessed by the prototype and the example [Smith and Medin, 1981].

Much of the trail-clearing cognitive research on prototypes and their relation to category structure has been done by Eleanor Rosch, who conducted a series of experiments documenting *prototype effects*, which are asymmetries in goodness-of-example ratings [Rosch and Mervis, 1975]. Such asymmetries could not exist under the classical view of category structure. Rosch found prototype effects along a large number of experimental dimensions: direct rating of goodness-of-example, reaction time, ease of example production, asymmetry in similarity ratings, asymmetry in generalization, and family resemblances. However, Rosch does not apparently believe that prototype effects mirror a particular category structure based on prototypes. Rather, she maintains that a variety of category structures and mental representations could account for the experimentally determined prototype effects. Nonetheless, some researchers believe that Rosch's experimental evidence does support a theory of representation for category structure [Lakoff, 1987]. Lakoff argues that domain knowledge is organized by humans in structures called *idealized cognitive models (ICMs)* and that prototype effects are side-effects of that organization.

Rosch's family resemblance measure is a primary example of a typicality measure, and it has been used to identify prototypes for classification tasks by Zhang [1992]. This typicality measure is based on the statistical distribution of instances in the class and on the distribution of features values within class and between classes. Zhang suggests that the average typicality of instances in a dataset can be used to characterize a dataset as a whole.

Work from philosophy, psychology and cognitive science thus converged to undercut theories of categorization that depended on the presence of necessary and sufficient membership criteria and substituted a view of concepts that depended on recognizing prototypical examples. Case-based reasoning is built in part on the general notion that examples — cases — are not fungible and that careful attention to individual cases is a basis for reasoning in a variety of applications [Rissland, 1977; Rissland, 1978a; Rissland, 1981] For instance, Rissland distinguishes several different categories of examples used to illustrate a theory: *start-up* examples, *model* examples (that are akin to the prototypes used here), *reference* examples, *anomalous* (or *pathological*) examples and *counter-examples* [1978b].

2.5.1.2 Facilitating Case Retrieval

In general, case-based reasoning (CBR) systems retrieve similar cases for further analysis or adaptation, not simply for classification [Rissland, 1989]. However, a number of case-based reasoning systems have relied on prototypes for memory organization. In these systems, a prototype is an index to cases that are similar to the prototype. Procedurally, prototypes give rise to a two-phase retrieval process. The first phase compares a new case to the small set of prototypes. Once the most similar prototype has been found, in the second phase, the new case is compared to the cases that are indexed under the prototype. We survey how prototypes have been selected by several CBR systems.

McCarty and Sridharan proposed a representation for legal cases as consisting of legal case prototypes plus possible deformations of them [1982]. Rissland investigated how examples are related in terms of how they are built from one another [1981]. This work laid a foundation for later work on *dimensions* and the creation of *hypotheticals* [Rissland *et al.*, 1984; Rissland and Ashley, 1986; Ashley, 1990]. These lines of research anticipated, partially from a jurisprudential perspective, Lakoff's view from cognitive science that some categories can be represented as a radial structure with a prototype at its hub. Rules generate other category members at the spokes, and still other category members conceptually reside along the rim [Lakoff, 1987].

More recently in the law, Sanders applied the notion of prototypical plans plus deformations in a program to structure transactions to minimize tax liability [1994]. She appeals to the time-honored legal idea of a *safe harbor* plan, a stereotypical plan to achieve a certain result under the tax code, as a prototypical transaction.

The ReMind CBR commercial development shell [Cognitive Systems, Inc., 1990] also incorporates a facility for the user to create prototypes to index a case base. Prototypes are combinations of simple predicates on features (e.g., bathrooms > 2 and schools = good). When a problem case satisfies these predicates and therefore matches a user-defined prototype, all the cases indexed under that prototype are retrieved for further filtering and analysis.

PROTOS is a CBR system that does rely on case prototypes for classification, the diagnosis of ear disease [Bareiss, 1989]. In PROTOS, prototypes emerge from having been matched to previous cases. Prototypes capture the typical features of a diagnostic category and are the cases to which appeal is first made. The similarity of an input case to the prototypes (the prototypicality rating) determines the order in which stored cases are selected for further, knowledge-based pattern matching. The degree of prototypicality of a case is increased for each example that is successfully matched to the prototype. PROTOS is an interactive assistant for intelligent classification and prototypicality ratings may be increased manually by a supervising teacher as well.

Thus in case-based reasoning, prototypes have been used in several systems as the cases to which comparison should be made first. Case-based systems have been built on the longstanding recognition that cases play different roles for different tasks. Prototypes may be viewed as a class of cases that are useful for classification. From this perspective, the creation of a case-base is analogous to the selection of prototypes to be stored in a nearest neighbor classifier, especially where the cases use the same representation scheme.

In the following section we outline the applications of prototypes in statistical pattern recognition and neural network algorithms that make predictions based upon neighborhoods of instances.

2.5.1.3 Local Learning Algorithms

The nearest neighbor algorithm is a member of a class of algorithms sometimes called *lazy learning* or *memory-based* methods, which do not process training data until a prediction needs to be made [Stanfill and Waltz, 1986; Atkeson *et al.*, 1996]. These methods are explicitly *local* methods, in that they use the instances in a neighborhood of a test case to make a classification or approximate a function [Bottou and Vapnik, 1992]. For example, (1-)nearest neighbor algorithm simply uses the class or function value of the nearest instance or point in $\Re^n$, real space of dimension n. By contrast, *locally weighted regression* fits a surface to points in a neighborhood of a test instance [Atkeson *et al.*, 1996].

Another type of local learning algorithm that relies on prototypes is the radial basis function (RBF) network [Poggio and Girosi, 1990; Hutchinson, 1993; Wasserman, 1993]. RBF networks are a function approximation method in which Gaussian or other basis functions are located at a set of prototypical points in the input space. Often these prototypical points are called *centers*, since they are the centers of the usually radially-symmetric basis functions. The function approximation at a new point is a weighted sum of the values of the basis functions at nearby centers, weighted by the distance of the new point from the centers.

One question that arises for all memory-based methods is how to select the instances to be stored. For RBF networks, this question is how to select the centers. While the details of RBF networks are not important here, there is a similarity between the problem of finding appropriate RBF centers and the problem of prototype selection for traditional nearest neighbor classification algorithms. Several approaches have been applied to the problem of placing the centers, including the computationally expensive technique of using all input instances as centers [Wasserman, 1993], invoking simple clustering methods [Burrascano, 1991], performing learning vector quantization (LVQ, described later in this section) [Kohonen *et al.*, 1988], and applying supervised learning algorithms [Wettschereck and Dietterich, 1992]. Wasserman [1993] observes that there is no guaranty of optimality for the cluster centers under any of the known techniques, and that cross validation is often used to select the best centers from among a set of candidates.

Moody and Darken [1989] suggest a hybrid neural network that uses unsupervised learning to find the centers and then uses supervised learning, with Gaussian radial basis functions located at those centers, to produce an output value. Their algorithm is best suited to learning continuous or piecewise continuous real-valued mappings [Moody and Darken, 1989]. The unsupervised learning component uses an *adaptive k-means clustering* algorithm for finding the centers. This incremental algorithm starts with a random set of centers. At each step, a random training vector is drawn, and the closest center is moved toward the training vector. A learning rate parameter governs how far the center is moved. Applying this hybrid technique to a time-series prediction task, Moody and Darken show comparable accuracy to backpropagation and estimate a speed-up of two to three orders of magnitude.

Vector quantization is another incremental technique for finding prototypes for unsupervised vectors of real numbers [Hertz *et al.*, 1991]. Kohonen and colleagues have extended vector quantization to supervised data with real-valued components in an algorithm called Learning Vector Quantization (LVQ) [Kohonen *et al.*, 1988; Kohonen, 1989]. For each training vector, the algorithm computes the closest prototype of a set of prototypes of a fixed cardinality. The closest prototype vector is moved toward the training vector if the classification is correct, and moves the closest prototype vector away from the training vector if the classification is incorrect. The locations of the other prototypes in the set are left unchanged. A more recent version of the algorithm (LVQ2) adjusts the decision boundary by moving both a misclassifying closest prototype and a correct next-closest prototype, but only under certain conditions. LVQ2 has shown improved performance over LVQ on some tasks [Kohonen *et al.*, 1988; Hertz *et al.*, 1991] .

Just as there is no guaranty for the optimality of the placement of centers for radial basis functions and local learning algorithms in general, there is no guaranty for the selection of prototypes for nearest neighbor classifiers. In the next section we describe algorithms designed to select prototypes as effectively as might be achieved.

2.5.2 Prototype Selection for Nearest Neighbor Classifiers

Reducing the number of prototypes used for nearest neighbor retrieval has been a topic of research in pattern recognition and machine learning for over 25 years [Hart, 1968]. The problem is sometimes called the *reference selection problem* and the algorithms to perform this task have been called *editing algorithms* or *editing rules* [Dasarathy, 1991]. Editing algorithms fall into two general groups. The first set of algorithms stems from pattern recognition research in the 1970's. The second set of algorithms has been presented more recently by machine learning researchers, primarily in the 1990's. While the algorithms presented by the two groups are not very different (and, in some cases, are identical), in view of the historical division of efforts in this area, we treat the two sets of algorithms in separate subsections. In-depth surveys of editing algorithms have been published by Aha [1990] and Dasarathy [1991].

As described by Dasarathy [1991], prototype selection algorithms have two goals, which are the two major goals of classification algorithms in general. The first goal is to reduce the computational expense of applying the nearest neighbor algorithm. Since the computational expense of applying a nearest neighbor algorithm can be great, this is a worthy goal. The second goal is to increase the accuracy of nearest neighbor algorithm predictions. Freund and Schapire have observed that, even *after* a seminal editing algorithm was used to reduce a set of stored prototypes (the Condensed Nearest Neighbor algorithm [Hart, 1968]), the remaining prototypes nevertheless showed a tendency to overfit the training data [1996].

Broadly speaking, there are three main approaches: (1) instance-filtering, (2) stochastic search, and (3) instance-averaging. By far, the greatest bulk of work falls into the instance-filtering camp. A rule is used incrementally to determine which instances to store as prototypes and which to throw away. Examples of this approach have included storing misclassified instances [Hart, 1968; Gates, 1972; Aha, 1990]; storing typical instances [Zhang, 1992]; storing only training instances that have been correctly classified by other training instances [Wilson, 1972]; exploiting domain knowledge [Kurtzberg, 1987]; and combining these techniques [Voisin and Devijver, 1987]. Stochastic search algorithms for prototype selection have been applied by Skalak [1994] and Cameron-Jones [1995]. Still other systems deal with prototype selection by storing averages or abstractions of instances [Chang, 1974; de la Maza, 1991].

Filtering Algorithms. Table 2.1 collects the major editing algorithms developed by researchers generally associated with pattern recognition research. We describe each algorithm in turn.

Table 2.1: Classical nearest neighbor editing algorithms.

Name	Abbrev.	Cite	Derivative of
Condensed NN	CNN	Hart, 1968	
Reduced NN	RNN	Gates, 1972	CNN
Iterative Condensation	ICA	Swonger, 1972	CNN
Edited NN	ENN	Wilson, 1972	
Selective NN	SNN	Ritter, 1975	
Unlimited ENN		Tomek, 1976	ENN
All k-NN ENN		Tomek, 1976	ENN
MultiEdit		Devijver, 1980	

Condensed Nearest Neighbor. Apparently the earliest editing rule was Hart's Condensed Nearest Neighbor algorithm [1968]. Hart uses a bottom-up approach, adding an instance to the prototype set if it is misclassified by applying the 1-nearest neighbor rule with all the other instances used as prototypes. Hart introduces the notion of a *minimal consistent subset*, a smallest subset of the complete set that also classifies all the original instances correctly, but his algorithm does not achieve such a subset. In practice, it does reduce the number of instances retained, however.

The Condensed Nearest Neighbor algorithm was tested on an artificial task and on a typewritten character recognition task. On the character recognition task, 197 prototypes were identified from 6295 training instances. An error rate of 1.28% was reported for the Condensed algorithm, but the error rate apparently was not reported for a nearest neighbor classifier using all instances as prototypes. The results were described by author as “disappointing” [Hart, 1968, p.516], in that other algorithms achieved a lower error rate at less computational expense.

Reduced Nearest Neighbor. The Reduced Nearest Neighbor algorithm is an iterative, top-down variant of the Condensed algorithm [Gates, 1972]. An instance is deleted from the prototype set if its deletion does not result in a misclassification of any of the other instances. Deletion continues until no further deletions have been made in an iteration.

The algorithm was tested on Iris data only [Gates, 1972]. In Gates’s experiment, the distribution of instances in each class was exactly the same and a single training set/test set split of the data was used. The training set consisted of 120 instances and the test set consisted of 30 instances. Fifteen prototypes were identified by the Reduced Nearest Neighbor algorithm, which gave 100% accuracy on the test set. The nearest neighbor algorithm with 120 instances stored also gave 100% accuracy.

Iterative Condensation. The Iterative Condensation algorithm (ICA) is also a variant of the Condensed algorithm, with the advantage that it may either add or delete prototypes from the prototype set [Swonger, 1972]. The prototype set is initialized with one instance from each observed class in the training set. The algorithm makes multiple passes through a training set. During each pass, ICA deletes prototypes from the prototype set that were not the closest prototype for any instance in the training set. Instances that have the highest *margin of misclassification* (the difference between the distance to the closest prototype of the correct class minus the distance to the closest prototype of an incorrect class) for a pass are added to the prototype set. The algorithm is not guaranteed to converge.

The Iterative Condensation Algorithm was applied to a single alphanumeric character recognition problem. The algorithm identified 87 prototypes from 867 instances. The algorithm yielded approximately a 5% error, and was described as showing a “moderately high level of recognition" [Swonger, 1972, p.516]. The error rate on this task apparently was not given for a nearest neighbor classifier that used all training instances as prototypes.

Edited Nearest Neighbor. The Edited Nearest Neighbor algorithm is a top-down algorithm that retains correctly classified instances in the prototype set [Wilson, 1972]. For each instance in the data set, the k-nearest neighbor rule is applied to determine if the instance is correctly classified according to that rule. (In this selection phase, k is fixed in advance, and $k = 3$ is suggested by Wilson.) If the instance is correctly classified, it is retained; if it is not classified correctly, the instance is deleted from the prototype set. Wilson's algorithm then uses a 1-nearest algorithm for the classification of new instances. The algorithm was not tested on any real task [Wilson, 1972].

Selective Nearest Neighbor. In the Selective Nearest Neighbor algorithm, Ritter and colleagues heuristically extend the definition of a minimal consistent subset [Ritter *et al.*, 1975]. In addition to the consistency and minimality requirements, Ritter introduces a requirement that each instance in the data set be nearer to a prototype of the same class than to *any* instance in the other class. This additional requirement gives rise to a relation that associates with each instance the instances that are of the same class and are closer than any instance of another class. The prototype set generated by the Selective algorithm is a smallest subset of the dataset that contains at least one related member for each instance in the dataset.

The algorithm was tested on 16 binary tasks that used mass spectra to distinguish chemical states. The algorithm identified an average of 149 prototypes from 430 training instances, which was the same number as the Reduced Nearest Neighbor algorithm and fewer than the average for the Condensed Nearest Neighbor algorithm (180 prototypes). On average across the 16 tasks, a nearest neighbor algorithm was 77.7% accurate, the Condensed algorithm 75.7% correct, the Reduced algorithm 75.2% correct, and the Selective algorithm 76.6% correct.

Unlimited and All k-NN Edited Nearest Neighbor. Tomek gives two variants of the Edited Nearest Neighbor algorithm (ENN) [Wilson, 1972; Tomek, 1976]. In the first variant, the Unlimited ENN algorithm, the ENN procedure is iteratively applied, with additional passes made through the set of stored prototypes. In the second variant, the All k-NN algorithm, the ENN procedure is repeated using an i-nearest neighbor algorithm in the selection phase for $i = 1, 2, \ldots, k$. The algorithm was tested only on artificial tasks.

MultiEdit. The MultiEdit algorithm [Devijver and Kittler, 1980] is an iterative top-down algorithm that discards misclassified instances from the prototype set, but invokes a form of cross validation to do the editing. The data are randomly partitioned into an ordered list of N subsets. Each of the N subsets is classified by a 1-nearest neighbor algorithm, using the instances in the next subset in the partition as prototypes. Instances that are misclassified are discarded. The remaining data are pooled to form a new set and the procedure is iterated. If a threshold number of iterations result in no further editing, the loop is exited and the algorithm outputs the remaining instances. The algorithm was tested on a single artificial two-class problem with a hyperbolic decision boundary.

Since the foundational work done around the 1970's with editing algorithms, researchers from machine learning have re-visited the prototypes selection problem. Table 2.2 lists notable recent examples. Many of the more recent nearest neighbor algorithms can work directly with symbolic attribute values and admit weighting of individual instances by scalars. Weighting instances by real values is a generalization of the 0-1 weights implicitly applied by algorithms that merely retain or delete an instance in the prototype set.

IB2 and IB3. Aha's IB2 algorithm is very similar to the Reduced Nearest Neighbor algorithm [Aha, 1990]. For symbolic predictions, IB2 saves only misclassified instances. In the case of numeric classes, only those instances are saved whose prediction is outside a parameterized interval around the actual classification. A second algorithm, IB3, uses only *acceptable*, non-noisy instances as prototypes. To determine if an instance is acceptable,

Table 2.2: Example machine learning nearest neighbor editing algorithms.

System	Cite	Stored Instances
IB2	Aha, 1990	misclassified
IB3	Aha, 1990	statistically reliable
PEBLS	Cost and Salzberg, 1991	weighting
EACH	Salzberg, 1991	weighting
TIBL	Zhang, 1992	typical
BIBL	Zhang, 1992	boundary
SRIBL	Zhang, 1992	misclassified

an instance is provisionally accepted if it is misclassified according to the (applicable symbolic- or numeric-class) criteria of IB2. Only a subset of those acceptable instances are ultimately retained and allowed to participate in predictions, however. This subset consists of each acceptable instance whose predictive accuracy is significantly higher on instances that are close to the acceptable instance than on all instances. To determine statistical significance, prediction records are maintained for stored instances. These records include the number of correct predictions and the total number of predictions attempted. A statistical test, the confidence intervals of proportions test, is applied to the prediction records to determine if each instance provides significantly more accurate classification. IB1 is Aha's implementation of a 1-nearest neighbor algorithm.

EACH and PEBLS. The EACH system [Salzberg, 1991] and PEBLS [Cost and Salzberg, 1993] associate a numerical weight with each instance in memory, which is used as a coefficient in the computation of the distance between instances. Like Aha's IB3, the weight reflects the classification performance history of each instance. The weight is the ratio of the number of times the instance was used as a classifying neighbor to the number of times it was used correctly. This weighting results in unreliable instances having weights that are greater than one, resulting in a greater weighted distance from unreliable instances.

TIBL, BIBL and SRIBL. Zhang developed three instance-based learning systems to compare the classification accuracy of storing misclassified instances (SRIBL), typical instances

(TIBL), and atypical boundary instances (BIBL) [Zhang, 1992]. SRIBL (Storage Reduction Instance-Based Learning System) follows IB2, repeating the process of finding misclassified instances and storing them until all instances are correctly classified. Zhang's TIBL (Typical Instance-Based Learning System) instead stores the most typical instance that is not currently stored whose addition to instance memory results in a correct classification for each previously misclassified instance. Zhang based the measure of typicality on Rosch's family resemblance measure, a ratio of an instance's intra-class similarity to its inter-class similarity. In TIBL, each stored exemplar is assigned a weight that reflects its typicality, so that the more typical instances have greater influence on a classification decision. The BIBL (Boundary Instance-Based Learning Algorithm) algorithm stores instances of the least typicality, which were exceptional and boundary instances. On four real-world data sets, three of which were drawn from medical applications, BIBL performed the worst of the algorithms compared.

Stochastic Search. We introduced a simple method for selecting prototypes and features for nearest neighbor classifiers through genetic algorithms [1993] and through random mutation hill climbing [1994]. Cameron-Jones has offered an approach a heuristic based on minimizing the encoding length of a string representing a set of classified and misclassified instances and that builds in part on our own approach [1995]. Cameron-Jones's approach has the theoretical advantage over our own work that the number of prototypes desired need not be pre-specified in advance. However, in practice, Cameron-Jones's results are comparable to our own in terms of accuracy and storage requirements. More recently, Cherkauer and Shavlik have investigated the use of a genetic algorithm for feature selection for simple decision trees [1996].

Instance-averaging. All of the above prototype selection algorithms output a subset of the training set, possibly coupled with weights. A different approach is exemplified by the research of Chang [1974], which creates artificial prototypes that are not members of the training set. The idea is to start with every instance in a training set as a prototype, and then successively merge any two nearest prototypes of the same class so long as the classification accuracy is not decreased. The PROTO-TO system of de la Maza [1991] also creates artificial prototypes, which de la Maza calls "augmented." Datta and Kibler define concept descriptions in terms of a list of ideal prototypes that do not appear in the training set [1995]. Case-based reasoning systems that rely on case merging to reduce the number of stored instances include PROTOS [Bareiss, 1989] and CABOT [Callan *et al.*, 1991]. Kibler and Aha [1988] have compared instance-averaging with instance-filtering techniques and found that while the two methods appeared to have equivalent classification accuracy and storage requirements, misclassified instances can be found in the prototype sets of instance-averaging algorithms. They judged instance-filtering algorithms more appealing for that reason.

Active Learning. Editing the training instances for a nearest neighbor classifier may be considered a specific type of *active learning* [Plutowski *et al.*, 1994]. A classifier that performs active learning selects its own training examples. For example, Plutowski and colleagues have shown that a small number of examples (fewer than 25) can be used to train a neural network to predict a difficult chaotic time series, the Mackey-Glass equation [Mackey and Glass, 1977; Plutowski *et al.*, 1994]. Results such as this also support the view that only a few training examples or prototypes may be required for accurate classification even for some hard problems.

Krogh and Vedelsby have applied an ensemble of neural networks to determine when a learner should ask the label of an example and use it for training [1995]. Their approach is that if the real-valued outputs of the members of an ensemble are very different for a given example, that example should be labeled and included in the training set. This approach is an extension of the query-by-committee algorithm for classification problems, which selects an input that maximizes disagreement among the "student" learners in a committee of learning algorithms [Seung *et al.*, 1992].

2.5.3 Limitations of Existing Prototype Selection Algorithms

None of the existing editing algorithms is guaranteed to find a minimal consistent subset. Therefore, as a theoretical matter, there is still room for fresh research on prototype selection, notwithstanding the attention that has been given to this topic. However, as a practical matter, for some datasets, one or more existing algorithms has resulted in a fairly small number of prototypes that can achieve a very good level of classification accuracy. For example, Aha's IB3 algorithm achieves 79% accuracy on the Cleveland heart disease data set [Murphy and Aha, 1994] while retaining only approximately 4% of the 303 instances [Aha, 1990]. Results such as this hint that a small number of prototypes will suffice on some data. In Table 2.3, we collect published accuracies of some recent editing algorithms by other authors. Different numbers of training and testing instances were used in these experiments, and the experimental design varied from researcher to researcher, yielding results that are not directly comparable.

The Explore algorithm, which relied on random mutation hillclimbing to select prototypes [Skalak, 1994], was generally the best performing of the several variations investigated [Cameron-Jones, 1995]. The Explore algorithm consistently retained only a small number of prototypes and attained very good accuracy relative to nearest neighbor and to other editing algorithms. Explore and the other algorithms developed by Cameron-Jones do have one clear theoretical advantage over our own algorithms: the number of prototypes need not be specified by a user.

Nevertheless, existing editing algorithms have an unfortunate search bias. Except for those algorithms that rely on instance weighting, existing algorithms either add an instance or remove it from the prototype set each time some triggering condition occurs. These algorithms change the *cardinality* of the set of prototypes without investigating other prototype sets of that cardinality. Further, in all examined algorithms except Swonger's Iterative Condensation Algorithm, the search is unidirectional in cardinality: the size of a prototype set is either increased or decreased at each step. (Unfortunately, Swonger's algorithm is not guaranteed to converge.) Thus the potential is present for inadequate search for prototype sets of a given, small cardinality. Algorithms with the potential to search more effectively the space of small sets of prototypes would be useful.

Secondly, existing editing algorithms will not be as useful for composite classifier prototype selection. The objective of previous research was to build a single minimal classifier of maximal accuracy (with some presumed tradeoff between size and accuracy). By contrast, our objective

Table 2.3: Published accuracy of machine learning prototype selection algorithms. Blank entries indicate that we could not find a value reported in the research literature for an editing algorithm. "dlM" indicates IB3 results reported by de la Maza [1991]; "C-J" indicates IB3 results reported by Cameron-Jones [1995].

Data	*Algorithm*	*Prototypes(%)*	*Prototypes(#)*	*Acc.*
Breast Cancer Ljubljana	IB3 (C-J)	1.9	4	58.0
	Explore	1.1	3	71.1
Breast Cancer Wisconsin				
Cleveland Heart Disease	IB1	100.0	212	75.1
	IB2	32.0	68	71.4
	IB3	3.9	8	78.4
	Explore	1.7	4	82.0
Diabetes				
Glass Recognition	IB3 (dlM)			45.5
	Proto-to			48.0
Hepatitis	Proto-to			79.9
	IB3 (dlM)			38.7
Iris Plants	IB3 (C-J)	9.5	11	93.6
	Explore	4.0	5	94.4
LED-24 Digit	IB1	100.0	1000	47.9
	IB2	60.1	601	43.7
	IB3	25.3	253	46.6
LED-7 Digit	IB1	100.0	200	71.6
	IB2	41.6	83	63.0
	IB3	20.1	40	72.5
	Explore	5.5	11	71.4
Lymphography				
Monks-2				
Promoter	1-NN	100.0	105	87.7
	PEBLS	100.0	105	95.3
Soybean				

is to construct several small nearest neighbor classifiers in order to use them as components in a composite classifier.

Since each existing editing algorithm can output only one set of prototypes for a given presentation of training data, it can be used to build only one classifier. Combining multiple copies of the same classifier into a composite classifier will not increase classification accuracy. Nevertheless, there may be *ad hoc* ways to alter or retrofit existing algorithms to produce multiple sets of prototypes, but we try an approach that is specifically tailored to the composite problem.

2.6 Summary

The primary technical question in this dissertation is how to select prototypes for nearest neighbor classifiers whose predictions are to be combined. A great deal of research bears on this question, although no published research has investigated this question on a variety of tasks. Nevertheless, enough work has been done on the general issues of classifier combination and prototype selection that it will be useful to extract a few key points to support our approach to the problem and to outline the assumptions that we make.

Homogeneous, nearest-neighbor components. Little work has been done on combining only nearest neighbor classifiers and eminent researchers have raised specific questions about the utility of nearest neighbor classifiers for various types of combination, including boosting and bagging [Drucker *et al.*, 1994a; Breiman, 1992]. Further research is needed to determine whether nearest neighbor classifiers are inherently unsuited for combination.

Prototype selection. Although prototype selection has been shown to increase accuracy and decrease storage and application costs for nearest-neighbor classifiers, the question has not been addressed as to how to increase accuracy through prototype selection for sets of nearest neighbor classifiers whose predictions are to be combined. In addition, existing selection algorithms have a search bias that does not promote extensive search for small sets of prototypes.

Coarse-hypothesis classifiers. Occam's Razor, efficiency considerations, and our own and others' experimental results suggest that "simple", coarse nearest neighbor classifiers that store only a few prototypes can be accurate in many applications. More research is required to determine the range of application of classifiers that store a very small number of prototypes.

Concise level-1 representation. We have collected a number of results to show that a small number of components have given the highest results obtained with a composite classifier [Breiman, 1992; Kwok and Carter, 1990; Battiti and Colla, 1994]. In several reported studies, as few as three classifiers were found to give the best results. Dietterich and colleagues [1995] have also raised the important open question of how to achieve concise level-1 representations for composite classifiers.

In summary, our review of research to date on classifier combination and prototype selection has revealed no attempts to increase accuracy by selecting prototypes for sets of classifiers whose predictions are to be combined. In the next chapter, we begin to map one route to a solution of this problem.

CHAPTER 3

NEAREST NEIGHBOR COMPOSITE CLASSIFIERS

In this chapter we introduce our approach to creating composite classifiers that satisfy the three design criteria introduced in Chapter 2: (1) component accuracy, (2) component diversity and (3) efficiency. We discuss the benefits of building composite classifiers with nearest neighbor components that store a small number of prototypes found through sampling the training set. The role of this chapter is to describe the elements of our approach that are common to the algorithms we propose for two major composite architectures: boosting and stacked generalization.

This chapter discusses several disparate topics. We begin with the characteristics of our approach and show how adopting these characteristics is a reasonable response to the current state of research on classifier combination. The data sets that are used for our empirical evaluation are described, along with the rationale for their selection. The standard nearest neighbor algorithm is described and evaluated to give a baseline for our results. We then detail the sampling algorithm applied to select the prototypes used by the component nearest neighbor classifiers. Next, we investigate several factors that influence the utility of sampling. Finally, we suggest a method for accelerating the testing of each sample. This method arises from an examination of the distribution of the accuracies of a large number of classifiers as they are applied to each training instance.

3.1 Characteristics of the Approach

Our approach has four general characteristics that have not been investigated adequately by previous research, either individually or together. These characteristics have been adopted in response to our design criteria and to the state of previous research into classifier creation and combination, as summarized in the final section of Chapter 2.

1. *Homogeneous, nearest neighbor component classifiers*
2. *Prototype selection through sampling*
3. *Coarse-hypothesis classifiers*
4. *Concise level-1 representation*

3.1.1 Homogeneous Nearest Neighbor Component Classifiers

This design characteristic has two pieces: (1) classifiers from only one model class are combined, and (2) that model class is the class of nearest neighbor algorithms.

Homogeneous Classifiers. While other research has focused on combining different types of classifiers, the focus in this dissertation is the combination of a collection of homogeneous classifiers.

The goal of an inductive learning algorithm is to create a hypothesis that generalizes the training data [Brodley, 1994]. Given a set of training data, a learning algorithm effects a search in the space of possible hypotheses that generalize the data. This search is often intractable, even for simple problems. One way to limit this search is to limit the space of hypotheses to be searched. In this research, we perform a limited search of the set of hypotheses in part by sampling nearest neighbor classifiers, whose predictions are to be combined. It has not been shown that it is necessary to combine classifiers from different model classes in order to achieve high accuracy. For example, Breiman's combination algorithm showed high accuracy even though the component regression models were quite similar to one another [1992] (Section 2.3.3). Brodley's MCS system showed that diverse classifiers are *sufficient* to create a composite classifier more accurate than any of its components on common used data sets [Brodley, 1994].

Here we investigate the complementary question of whether heterogeneous components are *necessary*. We show that we still achieve very good generalization accuracy on a variety of problems even though the component classifiers are drawn from one model class.

Combining heterogeneous classifiers does not necessarily satisfy the design goals, particularly the goal to make classifiers maximally diverse. Zhang, Mesirov and Waltz combined a statistical model, a neural network and a memory-based classifier to predict protein structure [1992]. Despite the application of three ostensibly different classification algorithms, the authors made a specific finding that the three classifiers "not only had very close overall prediction accuracy, their detailed predictions also agreed with one another much more than with the real structure" [Zhang *et al.*, 1992, p.1063]. An analysis by Brodley of the three model classes used by MCS (nearest-neighbor, decision-tree, linear discriminant function) showed that for a third of the data sets, at least one pair of classifiers agreed on a prediction for over 90% of the examples.

A side-effect of using homogeneous classifiers is that, since the component classifiers use the same internal concept representation, the potential exists for combining these concept representations, rather than merely combining the class labels output by each classifier. For example, a combining classifier might base its class prediction on the distances of the neighbors computed by each k-nearest neighbor component classifier. Where the component classifiers apply different and incommensurate internal representations, the combining classifier cannot make use of the components' internal representations, at least not nearly as easily.

For example, consider a hypothetical composite classifier that combines a rule-learning algorithm, a nearest neighbor algorithm and a backpropagation algorithm. Each algorithm creates hypotheses that use a different representation language. Difficulties are posed by two possible solutions to the problem of combining these representations. An algorithm that used the disparate representations of these algorithms either would have to (a) reduce all three representations to a single representation, or (b) be capable of using all three representations. An alternative solution would be to make the heterogeneous classifiers applicable in different regions of the instance space [Utgoff, 1989; Brodley, 1994], but that solution surrenders the potential to increase accuracy derived from having more than one component classifier classify each instance. While other options might also be available, the general problem of using the internal representations of the components would be ameliorated if the component classifiers all used a uniform representation for their concept hypotheses.

Nearest Neighbor Classifiers. Nearest neighbor algorithms are paradigmatic instance-based classifiers. Fifty years of research has shown them to be effective in a variety of domains. It has also been shown analytically that the nearest neighbor rule has an upper bound of twice the smallest generalization error theoretically obtainable, called the *Bayes error* [Cover and Hart, 1967]. Despite the demonstrated utility of nearest neighbor algorithms, there are few examples of combining component nearest neighbor classifiers in the research literature. Studying the combination of nearest neighbor classifiers will help fill a gap in our knowledge about the behavior of various model classes in combination.

Investigating composite classifiers with nearest neighbor components also allows us to address an important question raised by Breiman about the utility of combining nearest neighbor classifiers. Breiman has observed that nearest neighbor algorithms are *stable* in the sense that the hypothesis of a nearest neighbor classifier changes very little when small changes to the training set are made. Breiman reports that the stability of nearest neighbor classifiers distinguishes them from CART (decision) trees and some kinds of neural networks [1994a; 1994b]. He argues that stability makes nearest neighbor algorithms unsuitable for the bagging re-sampling technique to construct diverse classifiers that we discussed in Section 2.3.6. In our approach, we deliberately attempt to create *unstable* nearest neighbor classifiers by severely limiting the number of stored prototypes. Ironically, this instability may make these nearest neighbor classifiers *more* conducive to combination. Our approach could be called the *radical destabilization* of nearest neighbor classifiers.

3.1.2 Prototype Selection through Sampling

Prototype selection has been shown to be a useful technique for simultaneously increasing the accuracy and decreasing the computational cost of applying a single, independent nearest neighbor classifier. However, no work has exploited the benefits of prototype selection for nearest neighbor classifiers that are to work in combination.

In our review of prototype selection research to date, we observed that one shortcoming of existing algorithms is that they fail to search adequately for small sets of prototypes. To avoid this defect, we rely on random sampling of prototype sets of a small, fixed cardinality.

Random sampling is also a means to attain diverse component classifiers. Some measure of component classifier diversity may be achieved "for free" through stochastic methods. Previous research has shown the utility of random methods for creating component neural networks and decision trees, but not for creating instance-based classifiers. We will show that for many data sets, prototype sampling also has been shown to be an effective way to create a single nearest neighbor classifier [Skalak, 1994], and the extension to composite classifiers is a logical next research step.

Sampling prototype sets is tantamount to sampling nearest neighbor classifiers. Since we hold the other parameters of a nearest neighbor classifier fixed, a set of prototypes is in a 1-to-1 correspondence with a nearest neighbor classifier. Despite the attention historically devoted to nearest neighbor classifiers, little is known about the characteristics of the population of nearest neighbor classifiers. In general, sampling is a method for learning about the population without exhaustive enumeration and analysis.

We sample instances from the training set. An alternative approach might attempt to spawn prototypical instances by averaging or merging old ones, resulting in new, artificial instances. Unfortunately, artificial prototypes may not capture implicit constraints between feature values in real data. Further, while artificial prototypes may be useful to make class predictions, they may not have the *imprimatur* of real instances in their inherent explanatory power. An analogy can be made to citing legal cases: while it may be instructive and probative to cite a hypothetical case [Rissland, 1985; Rissland, 1990; Ashley, 1990], only *bona fide* cases may be cited in a legal brief as authority for a proposition of law. An additional complication for artificial instances is that it would mandate solving the problem of how to create prototypes where some of the features take symbolic or discrete values, which prohibit averaging feature values.

3.1.3 Coarse-Hypothesis Classifiers

In this dissertation we apply minimal nearest neighbor classifiers. These classifiers create class boundaries that partition the instance space into a small number of regions, one for each prototype. The class boundaries are usually only a coarse approximation to the true class boundaries. These classifiers may be considered coarse in a sense analogous to a *coarse* topological space in point-set topology. A topology on a set is a collection of subsets that satisfy certain properties. One topology $\mathcal{A}$ is *coarser* than a topology $\mathcal{B}$ if $\mathcal{A} \subset \mathcal{B}$ [Munkres, 1975].

Our reliance on coarse-hypothesis classifiers also can be seen as an attempt to use "high-bias" classifiers, as characterized by statistical research [Breiman, 1994b]. In classical regression problems, generalization error can be written as a sum of (squared) "bias" error and a "variance" error [Wonnacott and Wonnacott, 1972]. In a high-bias classifier, the bias component of the error is large on average; high-bias models tend to underfit a data set. For a recent discussion of statistical bias in the context of classification problems, see Kong and Dietterich [1995]. In this research we show that high-bias classifiers can be useful composite classifier components.

In Chapter 1, we discussed the substantive benefits of applying simple, coarse classifiers. Applying simple classifiers has methodological benefits, as well. Using classifiers that are easy to generate permits a form of empirical research that may better expose the structure of the space of classifiers under consideration. We can examine the structure through classifier sampling and experimentation. Several examples of this procedure can be seen in this dissertation. On many data sets, simple nearest neighbor classifiers exhibit accuracy nearly as high — and in some cases higher — than nearest neighbor classifiers that use all instances as prototypes. This result may well reflect overfitting of the training data caused by storing all the training instances as prototypes, but it is not yet known why such simple nearest neighbor classifiers work as well as they do on some commonly used data sets.

3.1.4 Concise Level-1 Representation

We use few component classifiers partly because we want to build efficient composite classifiers. We use few component classifiers also as an attempt to respond to Dietterich's call for composite classifiers that have a concise level-1 representation, more concise in particular than that typically achieved by error-correction output coding [Dietterich and Bakiri, 1995]. Since in stacked generalization there are as many attributes in the level-1 representation as there are components, fewer component classifiers means a more concise level-1 representation.

3.1.5 Summary

Our literature review has shown that where the question has been investigated for other classifier classes, combining a very small number of classifiers has led to increased composite accuracy over the most accurate component. We investigate whether this result holds for nearest neighbor classifiers as well.

We assume that a class prediction is the only information passed from a component to the combining classifier in a stacked generalizer. This assumption will be useful when a confidence level, concept hypothesis, class ranking or similar data are not output by the component classifiers. This assumption keeps our algorithms general, so that potentially they are applicable to a wider set of model classes.

One of the background assumptions of this work is that a strong domain theory is not available to inform the search for an appropriate combining classifier and complementary component classifiers. This is an important assumption, since using domain knowledge is often the key to improved system-building of all sorts. In general, more sophisticated approaches to selecting component and combining classifiers could be applied where a model of the domain is available, such as domain model available in the Integrated Processing and Understanding of Signals (IPUS) architecture of Lesser and colleagues [1995], or in the knowledge-based feature generation model of Callan [1993]. We search for component classifiers and a combination method that do not require extensive hand-coding of domain knowledge.

Our four-characteristic approach has been adopted, therefore, as one response to the composite classifier design criteria and to the state of previous research, including the open questions and problems raised by well-known researchers. This approach emphasizes the combination of small, sampled components, as opposed to the combination of more complex components that characterizes much previous research. Our approach therefore represents a fairly extreme approach to composite classifier construction. We have adopted these constraints to explore the boundaries of approaches to classifier combination, to determine better where the limitations and the potential lie.

3.2 Experimental Design

We desire that our composite architectures be applicable to a wide variety of data sets. In this section, we describe the data sets used in our experiments and the rationale for their selection.

3.2.1 Data Sets

Thirteen data sets were selected to provide coverage of many of the dimensions on which data sets vary, such as the number of features, number of values of features, types of feature values, number of irrelevant features, number of cases, level of attribute and class noise, frequency of missing values, data set density, number of classes, default accuracy, and difficulty [Zheng, 1993]. The data we use for experimental comparisons were drawn from the University of California at Irvine (U.C.I.) Repository of Machine Readable Databases [Murphy and Aha, 1994]. Ten data sets were chosen from the representative data sets for the U.C.I. Repository identified by Zheng. We selected three data sets (Breast Cancer Ljubljana, Cleveland Heart Disease and Iris Plants) because previous results have revealed that stacked generalization may not improve accuracy on them [Ali and Pazzani, 1996]. We have provided an overlap with the data sets used in recent machine learning dissertation research with hybrid classifiers, as by Brodley [1994] and Ali [1996].

1. **Breast Cancer Ljubljana.** The task is to determine whether breast cancer will or will not recur. The suppliers of the data have asked users to recite that the data were obtained originally by the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data.

2. **Breast Cancer Wisconsin.** The task is to determine whether breast cancer will be malignant or benign. This database was obtained originally from the University of Wisconsin Hospitals, Madison, from Dr. William H. Wolberg [Mangasarian and Wolberg, 1990].

3. **Cleveland Heart Disease.** The task is to predict from a patient diagnosis whether heart disease is or is not present. This heart disease data set comes in two-class and five-class versions; we use the two-class version.

4. **Diabetes.** The task is to decide whether an individual has the symptoms of diabetes, based on World Health Organization criteria. The data were collected from a population of Pima Indians near Phoenix, Arizona.

5. **Glass Recognition.** The task is to identify which one of the six types of glass is present from chemical elements in a sample. The examples were collected by B. German of the Home Office Forensic Science Service, Aldermaston, Reading, England.

6. **Hepatitis.** The task is to predict from test results and diagnostic examination whether a patient will live or die from hepatitis. This data set is missing 167 attribute values. Seventy-five instances are missing one or more values.

7. **Iris Plants.** The task is to classify iris plants into one of three iris plant varieties. This data set is the classic one due to Fisher [1936].

8. **LED-24 Digit.** The task is to determine which of the digits 0-9 is represented by the binary features for a seven-element light-emitting diode (LED). The task is the same as for the LED-7 data set (below), with the additional complication that 17 irrelevant binary attributes are included in each instance description. Each of the seven relevant features also has a 10% chance of having its value toggled (from 0 to 1 or from 1 to 0). Noise is present in both the training and test cases.

9. **LED-7 Digit.** The task is to determine which of the digits 0-9 is represented by the binary features for a seven-element light-emitting diode. Each of the seven binary features has

a 10% probability of having its value toggled. Noise is present in both the training and testing cases. This data set is not the version used by Breiman and colleagues [1984], for which the Bayes optimal accuracy is 74%.

10. **Lymphography.** The task is to categorize each patient-instance into one of four classes corresponding to lymph-node cancer diagnoses. The instance features are diagnostic features for potential lymph node malignancies. The lymphography data were obtained from the University Medical Centre Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data.

11. **Monks-2.** This artificial domain from Carnegie-Mellon University was used as the basis for the first formal international comparison of learning algorithms [Thrun *et al.*, 1991]. The task in the Monks-2 problem is to identify when exactly two of the features are "1". Each attribute takes a symbolic value from a set of two, three or four symbolic values. The data for the competition were divided into a single test and a single training set, but we have used a 10-fold cross-validation partition, for uniformity with the rest of our data sets.

12. **Promoter.** The task domain is E. Coli promoter gene sequences (DNA), and the task is to classify gene sequences as "promoters" or "non-promoters" based on 57 sequential DNA nucleotide positions.

13. **Soybean.** The task is to categorize soybean diseases, based on a set of categorical attributes, into one of 19 classes. This data set is commonly called the "large" soybean data set.

Data sets can be divided into the following categories: those that contain instances with all numeric features, those with all symbolic features, and those with a combination of numeric and symbolic features (Table 3.1). Some of the U.C.I. data sets contain numeric encodings of symbolic values (e.g., 1 for red, 2 for green, 3 for blue), and we treat these as symbolic values. Binary features are considered symbolic Boolean values, as in the two light-emitting diode

(LED) data sets, where 0 represents "off" and 1 represents "on". We treat ordinal encodings that reflect an ordinal scale as numeric values. For example, the age of a patient might be encoded as a digit from 0 to 9, to represent age within ten-year intervals: 0 for $(0, 10]$, 1 for $(10, 20]$, and so forth.

Table 3.1: Summary of data sets used in experiments. In the column labeled "Feature Types", "N" designates numeric features; "S" designates symbolic features; "B" designates binary features.

Name	*Instances*	*Omitted*	*Features*	*Feature Types*	*Classes*	*Missing Values*
Breast Cancer Ljubljana $_{280}$	286	6	9	6S,3N	2	0
Breast Cancer Wisconsin $_{690}$	699	9	9	N	2	16
Cleveland Heart Disease $_{300}$	303	3	13	7S,6N	2	6
Diabetes $_{760}$	768	8	8	N	2	0
Glass Recognition $_{210}$	214	4	9	N	6	0
Hepatitis $_{150}$	155	5	19	13S,6N	2	167
Iris Plants $_{150}$	150	0	4	N	3	0
LED-24 Digit $_{200}$	200	0	24	B	10	0
LED-7 Digit $_{500}$	500	0	7	B	10	0
Lymphography $_{140}$	148	8	18	15S,3N	4	0
Monks-2 $_{430}$	432	2	6	S	2	0
Promoter $_{100}$	106	6	58	S	2	0
Soybean $_{300}$	307	7	35	S	19	712

3.2.2 Experimental Method

In this section, we describe the experimental techniques used in the experiments in this dissertation. To estimate the accuracy of the classifiers produced by each algorithm, we performed 10-fold cross-validation. In ten-fold cross-validation, the data set is partitioned into ten disjoint subsets. In each fold, or "ply", one subset is held out as an independent test set and the remaining instances are used as the training set. A learning algorithm is then trained on the training set and tested on the test set.

Each data set was given a default partition that was used for every experiment with that data. Using the same partition assured that the relative performance of two learning algorithms is not due to different partitions of the data set. To ensure the potential application of any statistical significance test that might require equal-sized cross-validation plies, our partitioning algorithm yields ten subsets of equal cardinality. Therefore, a small, randomly selected remainder of instances (fewer than ten) may not appear in any ply. (For example, for the Cleveland Heart Disease data, which has 303 instances, there are 10 subsets of 30 instances each, and three instances are left over that do not appear in any subset.) The instances omitted from each cross-validation partition are listed in Appendix A. The subscript attached to each data set name in a table is the total number of instances appearing in the ten-fold cross-validation partition.

We used a one-tailed t-test for matched pairs to determine the statistical significance of the difference in the mean generalization accuracy of the two algorithms being compared. Ten-fold cross-validation was employed for all data sets. Unless stated, results are the percentage of test instances that are classified correctly.

For each experiment we report the average accuracy on the test set and its standard deviation across the ten plies. Our usual goal is to maximize test-set (generalization) accuracy. We assume the standard 0-1 loss function used generally for classification experiments, which assumes that every misclassification is equally serious [Ripley, 1996]. In general, our algorithms have obvious extensions to other loss functions, such as those that provide greater penalties for making errors on instances from one class.

3.3 A Nearest Neighbor Algorithm

In general, k-nearest neighbor algorithms make a prediction based on the k closest instances to an instance, according to a distance function that measures the similarity of two instances [Duda and Hart, 1973]. A smaller distance entails greater similarity. Given an instance whose class is unknown, the k-nearest neighbor algorithm predicts the most frequent class assignment among the closest k instances. The distance function used in our nearest neighbor computation computes an unweighted sum of feature differences, as we describe formally below. We assume that attributes may take binary, numeric or symbolic values. Thorough investigations of the characteristics of nearest neighbor algorithms have been done elsewhere, such as their sensitivity to changes in k and to different distance metrics [Aha, 1990; Wettschereck, 1994].

We rely on 1-nearest neighbor classifiers, rather than k-nearest neighbor classifiers for $k > 1$, because we radically limit the number of prototypes that are stored. Typically, only one prototype per class is stored. Where only a small number of prototypes are retained, the k nearest neighbors generally will include some prototypes that are quite distant from the test case. In particular, where only one prototype per class is stored, a k-nearest neighbor classifier would select one closest prototype from each of k classes, and a tie will result when the closest instances vote for a prediction. In the extensive StatLog project comparison, 16 classification algorithms were tested on 12 large, real-world data sets. For 75% of the data sets, it was found that the best results for a k-nearest neighbor classifier were attained for $k = 1$ [Feng *et al.*, 1993].

Pre-processing. As with many nearest neighbor algorithms, data are pre-processed (1) to complete any missing attribute values and (2) to scale numeric values [Dasarathy, 1991]. Attributes with missing values are given imputed values so that comparisons can be made between every pair of instances on all features. Scaling is done to place attribute values that have different absolute values within the same interval, or scale. Pseudocode is given in Figure 3.1.

a. Missing Values. In this pre-processing step, missing feature values for numeric attributes are instantiated with the median value of that attribute across all training instances. Missing feature values for symbolic attributes are assigned the mode value for that feature across all training instances.

b. Scaling. All numeric attribute values are linearly scaled to the interval $[0, 1]$. Extreme numeric feature values are squashed by giving a scaled value of 0 (1) to any raw value that is less (greater) than three standard deviations from the mean of that feature computed across all training instances. We use this approach rather than, say, standard normal form, in order to limit the effect of extreme, outlying values. The ReMind case-based reasoning development shell has incorporated a similar data pre-processing method [Cognitive Systems, Inc., 1992].

We can perform this scaling computation regardless of whether the values of the attribute are normally distributed. As in the transformation of a variable to standard normal form, however, we are implicitly assuming that its values are normally distributed. This assumption may or may not be correct on a given set of data, but other research with nearest neighbor classifiers has also assumed (at least to the same extent as we do) that the values of numeric attributes are distributed normally. Aha states (but does not show) that for the data sets used in his thesis [1990, p.119], the numeric attributes are not normally distributed, in general. However, for the Cleveland Heart Disease data, normalization using standard normal form performed the best of three normalization techniques. (Cleveland Heart Disease is the only data set in our collection that was also used in Aha's normalization experiments on four data sets.) Brodley used standard normal form for the representation of numeric attributes for the nearest neighbor classifiers incorporated in MCS [1994].

Similarity Computation. To compute the similarity between two scaled instances with all attribute values present, we use the Manhattan ("city block" or l_1) distance metric. The prototype with the smallest such distance to a test instance is its nearest neighbor (Figure 3.2).

Thus, if $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ are instances that have been pre-processed, F_n is the set of numeric features, F_s is the set of symbolic features, then the distance between x and y, $d(x, y)$ is

$$d(x, y) = \sum_{i \in F_n} |x_i - y_i| + \sum_{j \in F_s} d_s(x_j, y_j)$$

where d_s is an exact-match metric for symbolic features:

$$d_s(x_j, y_j) = \begin{cases} 0 & \text{if } x_j = y_j \\ 1 & \text{otherwise} \end{cases}$$

If there is a distance tie, the first such tied instance in a fixed ordering is used as the closest neighbor. The ordering of instances for U.C.I. data sets is the same as in the presentation of the instances in the files in that library [Murphy and Aha, 1994]. As usual, an instance is considered correctly classified by a prototype set if the instance's class equals the class of the prototype that is its nearest neighbor in the prototype set.

Key:

D : a set of data instances, both training and testing
T : a set of training instances
F_n : set of numeric features
F_s : set of symbolic features
μ_f : mean value over all instances in T
  of a numeric feature f
ν_f : median value of all instances in T
  of a numeric feature f
m_f : mode over all instances in T
  of symbolic feature f
σ_f : standard deviation over all instances in T
  of a numeric feature f
x : an instance in D
$f(x)$: value of feature f in instance x

```
procedure preprocess (D)
;; complete missing values for numeric features
loop for f in F_n
   loop for x in D
      when missing(f(x))
      f(x) ← ν_f
;; complete missing values for symbolic features
loop for f in F_s
   loop for x in D
      when missing(f(x))
      f(x) ← m_f
;; scale numeric values
loop for f in F_n
   loop for x in D
      f(x) ← scale(f(x))

procedure scale (f(x))
upper ← μ_f + 3σ_f
lower ← μ_f − 3σ_f
if f(x) ≤ lower
   f(x) ← 0
else if f(x) ≥ upper
   f(x) ← 1
else
f(x) ← (f(x) − lower)/(upper − lower)
```

Figure 3.1: Pseudocode for data set preprocessing.

Key:

```
P:  a set of prototypes
d:  the distance metric, as defined in the text
x:  a pre-processed data instance
procedure classify-by-1-nearest-neighbor (P, d, x)
smallest-distance ← *large-number*
closest-prototype ← nil
loop for p ∈ P
   when d(p, x) < smallest-distance
      smallest-distance ← d(p, x)
      closest-prototype ← p
finally return class(closest-prototype)
```

Figure 3.2: Pseudocode for 1-nearest neighbor classification algorithm.

3.3.1 Evaluation of Baseline Nearest Neighbor Classifier

This experiment provides the baseline accuracy for the 1-nearest neighbor algorithm. The results are given in Table 3.2.

Table 3.2: Percent test accuracy and standard deviation for baseline 1-nearest-neighbor algorithm.

Data	*NN*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8
Cleveland Heart Disease $_{300}$	77.0 ± 5.5
Diabetes $_{760}$	68.7 ± 5.1
Glass Recognition $_{210}$	73.3 ± 10.3
Hepatitis $_{150}$	77.3 ± 11.4
Iris Plants $_{150}$	91.3 ± 7.1
LED-24 Digit $_{200}$	36.5 ± 9.4
LED-7 Digit $_{500}$	71.2 ± 7.7
Lymphography $_{140}$	80.7 ± 8.9
Monks-2 $_{430}$	48.8 ± 4.7
Promoter $_{100}$	79.0 ± 9.9
Soybean $_{300}$	91.0 ± 3.9

This nearest neighbor implementation shows generalization accuracy in keeping with other implementations and with many other algorithms [Aha, 1990; Thrun *et al.*, 1991; Brodley, 1994]. For all but one data set, there is no reason to believe that nearest neighbor algorithms perform significantly worse on these data sets than most other learning algorithms.

The Monks-2 data does show low nearest neighbor accuracy. In fact, the nearest neighbor algorithm performs with less than default accuracy, which is approximately 67.1%. On the other hand, in a competitive survey of learning algorithms on this data, at least eight learning algorithms failed to beat default accuracy, including some of the best known decision-tree algorithms [Thrun *et al.*, 1991]. (Further, evaluations of statistical significance were not published and so it cannot be determined if the small increases in accuracy attained by some of the algorithms are significant. The survey did not include a nearest neighbor algorithm.)

However, many rule induction, neural network and decision tree algorithms performed with much higher accuracy than we have attained with the 1-nearest neighbor algorithm [Thrun *et al.*, 1991]. Low accuracy is in part due to the unsuitability of nearest neighbor to the task, which is akin to k-of-n tasks: identify instances where k bits of an n-bit binary string are 1. The Monks-2 task is to identify instances in which exactly two of the six features have the value 1. These instances are given class 1; the instances with class 0 do not have two 1's. The attributes are treated as assuming symbolic values. The problem is that — unlike other models such as decision trees or many neural networks — 1-nearest neighbor is very poorly suited to the task. The nearest neighbor of each instance of class 1 (which has two 1's) is frequently of class 0, and so instances of class 1 are rarely correctly classified by a 1-nearest neighbor algorithm. This task is similar to parity in that, with some regularity, an instance's nearest neighbors have a different class from the instance. In addition, the order of the instances in the data set also influences the performance of the nearest neighbor algorithm. In the event of a distance tie for the nearest neighbor to a test instance, this implementation takes the class of the instance that appears first in the data file. (Since this ordering is available to anyone retrieving the standard data set, this tie-breaking policy helps to ensure that an experiment may be repeated.)

3.3.2 Disadvantages of Some Nearest Neighbor Algorithms

Like any model class, k-nearest neighbor classifiers have learning biases and default implementations that make them inappropriate for some types of data. A list of objections to k-nearest neighbor algorithms includes the following:

1. Slowness and computational expense of classification
2. Usually trivial ($l_p, p = 1, 2$) similarity metrics, not informed by the problem at hand
3. Voting used to combine the classes of the nearest k instances
4. Uniform neighborhood shape (spherical) regardless of instance location
5. Uniform weight given to all features, instances and neighbors
6. Lack of a principled way to choose k , except through cross-validation or similar computationally expensive technique.

While many of the disadvantages can be addressed by more sophisticated variations on the nearest neighbor algorithm [Aha, 1990; Dasarathy, 1991; Wettschereck, 1994; Atkeson *et al.*, 1996], we use the standard version described above. Our ultimate goal is not to improve the nearest neighbor algorithm, but to improve classification by combining nearest neighbor classifiers. Despite their disadvantages, nearest neighbor algorithms have been used in real projects, such as for the prediction of power supply demands by an upstate New York power company, Niagara Mohawk [Aha, 1990]. Many current information retrieval systems rely on a vector-space model that uses a nearest neighbor algorithm to assess document similarity [Salton, 1989]. Nearest neighbor algorithms have also been applied to many domains over the last 50 years and have been performed comparably to allegedly more sophisticated algorithms on many tasks [Thrun *et al.*, 1991].

3.4 Classifier Sampling

Sampling is a statistical method that is usually invoked to estimate the value of a population statistic, such as the mean or median, by examining a proper subset of the population [Hansen *et al.*, 1953; Guttman *et al.*, 1971]. Here, we sample from a population of nearest neighbor classifiers. Our immediate goal is not to learn something about the population, but to find a nearest neighbor classifier that is simple and accurate. Our motivation for sampling diverges from the classical one; here, sampling provides a way to generate classifiers, as one-half of a generate-and-test algorithm [Barr *et al.*, 1981].

This section describes a generate-and-test algorithm for creating an independent nearest neighbor classifier. In later sections, we shall rely on the same algorithm for generating component classifiers. The generate-and-test algorithm relies on a sampling plan that samples coarse nearest neighbor classifiers by sampling small sets of prototypes [Skalak, 1994]. There is a one-to-one correspondence between a set of prototypes and a nearest neighbor classifier, since we have fixed all the other elements of the nearest neighbor rule (k , the distance metric, etc.) We invoke prototype sampling in part because it provides a way to target the search on small sets of prototypes, a characteristic not found in traditional editing algorithms (Section 2.5.3).

Technically, the sampling algorithm that we use is *stratified random sampling* [Hansen *et al.*, 1953; Thompson, 1992]. Whenever sampling is constrained to select one or more instances from each sub-group — from each *stratum* — of a heterogeneous population, the sampling is said to be *stratified.* In general, stratified sampling is used to assure that representatives of each stratum are taken, and to improve the accuracy of the desired population estimate for a given sampling cost. Where random sampling is used within each stratum, the technique is called stratified *random* sampling [Hansen *et al.*, 1953; Thompson, 1992]. The presence of instances with different class assignments makes our data sets heterogeneous populations, suitable for stratified random sampling.

3.4.1 Algorithm

As input, the prototype sampling algorithm takes the training set T and four other parameters: φ (the fitness function to determine the usefulness of a set of prototypes) k (k-nearest neighbors), p (the number of prototypes sampled from each class), and m (the number of samples taken). These parameters are fixed in advance. Unless otherwise specified, for all the experiments presented, φ is the leave-one-out cross-validation accuracy of a nearest neighbor classifier on the training set, $k = 1$, $p = 1$ and $m = 100$. The choice of $m = 100$ is discussed in Section 3.4.2.1. Thus, the number of prototypes sampled is the number of classes (s) "exposed" in the training set T . The classes *exposed* in the training set are those classes for which instances of that class are present in the training set. (If not all the possible class targets are represented in the training set, the data set classes are said to be *partially exposed* [Dasarathy, 1980].) The classifiers are minimal in the number of prototypes, since only one prototype is sampled from each class.

The sampling algorithm can be summarized as follows. Pseudocode is given in Figure 3.3. We assume that a training set has been identified.

1. Select m random samples, each sample with replacement, of p instances from each class exposed in the training set.

2. Store each sample in a nearest neighbor classifier. To evaluate the utility of that classifier, compute the value of a *fitness function* on that sample. The *fitness* of a sampled classifier is its classification accuracy on the training set using a 1-nearest neighbor rule and leave-one-out cross-validation.

3. Return the classifier with the highest classification accuracy on the training set. In the event of an accuracy tie, return the tied classifier that was generated first.

Prototype sampling is very simple to implement, and it runs in time $O(mps|T|q)$, where q is the number of attributes and s is the number of classes. For each of the m samples, the distance between each of the $p \cdot s$ prototypes and each of the training set instances from T must be computed. The distance computation for each pair of instances requires that the inter-feature distance be computed for each of the q features. Thus this algorithm is potentially very expensive. In order to keep the runtime down, we try to limit these parameters to small values, where possible. In the next section, we show in particular that a fairly small number of samples, m, need be taken to get good accuracy.

We focus on minimal nearest neighbor classifiers that store a minimal number of prototypes: one per class. So, for example, in the Breast Cancer Ljubljana data set, where there are two classes corresponding to a recurrence or non-recurrence of breast cancer, there are only two prototypes in each sample. One prototype will be selected from among patients for whom breast cancer did recur, and one prototype will be selected from among patients for whom

Key:

```
T :  training set
S :  set of classes exposed in  T ,  S = {S_i | i = 1, ..., s}
T_{S_i} :  training instances of class  S_i
P_j :  a set of prototypes
C_j :  a nearest neighbor classifier
φ :  fitness function,  φ(C_j, T)  is the accuracy of  C_j
  when applied to  T
p :  number of prototypes from each class
m :  number of samples of prototype sets
k :  cardinality of neighborhood ( k-nearest neighbor)

procedure sample-nearest-neighbor-classifiers (T, φ, k, p, m)
best-training-accuracy  ← 0.0
best-classifier  ←  nil
;; for each sample
loop for  j  from 1 to  m
   loop for  S_i  in  S
      ;; for each class, select prototypes
      append select-random-instances (p, T_{S_i})
      into  P_j
   C_j  ←  create-nearest-neighbor-classifier (P_j, k)
   ;; the fitness function is training set accuracy
   accuracy  ← φ(C_j, T)
   when accuracy  >  best-training-accuracy
      best-training-accuracy  ←  accuracy
      best-classifier  ← C_j
finally return best-classifier
```

Figure 3.3: Pseudocode for sampling nearest neighbor classifiers by stratified random sampling of prototypes.

the cancer did not recur. In such two-class data sets, these two classes may be viewed as corresponding to "positive" and "negative" examples. In multi-class data sets there is no sense of positive and negative examples. For example, in the Glass Recognition data set, there are six classes and therefore six prototypes will be selected. There, the classes correspond to the source and manufacturing process of glass fragments for crime scene analysis: building windows–float processed, building windows–not float processed, car windows, containers, tableware and headlamps. (Data sets were described in Section 3.2.1.)

Evaluating a set of prototypes based on training accuracy relies on an assumption common to many machine learning algorithms, that (out-of-sample) performance on a test set, *generalization accuracy*, will reflect performance on the training set. (Alternatively, the assumption is that test set performance will reflect performance on a "validation set," which is disjoint from both the training and the test sets. A validation set is typically used to set parameters or to determine when to terminate training.) This prototype sampling algorithm has at least two closely related infirmities that are endemic to many learning algorithms. First, a prototype set with higher training accuracy may have lower test accuracy than a sample with lower training accuracy. Higher training accuracy is sometimes evidence of overfitting. Second, a prototype set with lower training accuracy may show higher test accuracy. In this situation, the algorithm fails to recognize that a prototype set is superior out-of-sample.

3.4.2 Evaluation

In Table 3.3 we give the classification accuracy of the baseline nearest neighbor algorithm and that of a sampling technique that stores only one prototype per class. One hundred samples were taken. The sampling algorithm was run 20 times; average test accuracies are given.

The minimal nearest neighbor classifier significantly ($p \leq 0.05$) beats a full nearest neighbor classifier on three of the 13 data sets (Breast Cancer Ljubljana, Diabetes and Hepatitis). Given the tremendous savings in the cost of applying a minimal nearest neighbor classifier and the coarseness of the hypothesis space, it may be surprising that prototype sampling is *ever* significantly more accurate. A t-test reveals that the classifiers created by sampling are

Table 3.3: Classification accuracy of prototype sampling (PS) algorithm, one prototype per class.

Data	*Prototypes*	*NN*	*PS*	*Sig.*
Breast Cancer Ljubljana $_{280}$	2	67.5 ± 9.6	73.2 ± 8.2	0.019
Breast Cancer Wisconsin $_{690}$	2	96.2 ± 1.8	96.4 ± 2.4	0.422
Cleveland Heart Disease $_{300}$	2	77.0 ± 5.5	78.5 ± 8.0	0.298
Diabetes $_{760}$	2	68.7 ± 5.1	73.9 ± 4.1	0.007
Glass Recognition $_{210}$	6	73.3 ± 10.3	60.0 ± 8.5	1.000
Hepatitis $_{150}$	2	77.3 ± 11.4	82.7 ± 8.3	0.010
Iris Plants $_{150}$	3	91.3 ± 7.1	94.7 ± 5.5	0.053
LED-24 Digit $_{200}$	10	36.5 ± 9.4	32.2 ± 5.6	0.875
LED-7 Digit $_{500}$	10	71.2 ± 7.7	73.2 ± 7.8	0.095
Lymphography $_{140}$	4	80.7 ± 8.9	74.6 ± 5.2	0.968
Monks-2 $_{430}$	2	48.8 ± 4.7	46.6 ± 5.0	0.802
Promoter $_{100}$	2	79.0 ± 9.9	64.5 ± 10.4	0.996
Soybean $_{300}$	19	91.0 ± 3.9	65.8 ± 7.3	1.000

statistically indistinguishable on six other data sets (Breast Cancer Wisconsin, Cleveland Heart Disease, Iris Plants, LED-24 Digit, LED-7 Digit and Monks-2), and show significantly worse results on four data sets (Glass Recognition, Lymphography, Promoter and Soybean).

Thus, on nine of thirteen benchmark data sets, a classifier that (i) is a minimal member of its model class (when measured by the number of prototypes) and (ii) is generated by a naive sampling algorithm that drew a small number of samples (when compared to the population size) has accuracy higher than or statistically comparable to a classifier that stores all available instances as prototypes. No algorithm has shown better results on any of these data sets for an equal or smaller number of prototypes. A smaller number of prototypes would misclassify all the members of at least one class.

For reference, Table 3.4 gives the time to run the prototype sampling algorithm and to apply the resulting nearest neighbor classifier to the entire data set. (The data sets that have many classes show higher sampling costs: LED-24 Digit, LED-7 Digit and Soybean. This expense is a function of our particular implementation and is not an inherent cost of the algorithm.)

The primary focus throughout this dissertation is on nearest neighbor classifiers that incorporate only one prototype per class. There are several reasons for limiting our focus to

Table 3.4: Time to sample 100 sets of prototypes, select the most accurate sample (*Sample*) and classify the data set using that sample (*Application*), compared to the time to apply the nearest neighbor algorithm to the data set (*NN*). Values are wall-clock times, in seconds.

Data	*Sample*	*Application*	*Total*	*NN*
Breast Cancer Ljubljana $_{280}$	20.1	0.2	20.3	25.5
Breast Cancer Wisconsin $_{690}$	40.0	0.4	40.5	108.0
Cleveland Heart Disease $_{300}$	26.5	0.2	26.8	36.3
Diabetes $_{760}$	38.2	0.4	38.6	118.2
Glass Recognition $_{210}$	33.5	0.3	33.9	10.8
Hepatitis $_{150}$	15.4	0.1	15.5	10.9
Iris Plants $_{150}$	7.8	0.0	7.9	3.1
LED-24 Digit $_{200}$	92.4	0.9	93.4	17.7
LED-7 Digit $_{500}$	119.4	1.3	120.7	56.9
Lymphography $_{140}$	26.2	0.3	26.5	9.2
Monks-2 $_{430}$	21.6	0.2	21.9	42.3
Promoter $_{100}$	26.6	0.2	26.9	13.8
Soybean $_{300}$	400.7	4.4	405.1	63.7

these minimal nearest neighbor classifiers. First, we wish to study the most extreme case of nearest neighbor classifiers that use as few prototypes as possible, since the opposite extreme — storing all instances as prototypes — has been studied in depth.

Second, when the prototype sampling algorithm is used to select prototypes, the computational expense of finding the set of prototypes that displays the highest training accuracy increases linearly with the number of prototypes selected. For example, as the number of prototypes per class is increased from one to two, the computational expense is doubled, since the number of inter-case distance computations required to determine the accuracy of a prototype set is also doubled. In addition, since the search space is larger, one may also wish to extract a larger number of samples if more prototypes are selected in each sample.

Third, one of our design goals to emphasize diversity in the level-0 component classifiers. In particular, where the task is boosting a full nearest neighbor classifier, our design goal requires that we find component classifiers that make as few errors in common with that full nearest neighbor classifier. We assume that the maximal degree of diversity is to be found where only one prototype per class is used. As the number of prototypes is increased, we expect diversity to decrease. This assumption is consistent with Breiman's observation that a nearest neighbor classifier that stores many prototypes is stable.

Fourth, we can test empirically the accuracy of nearest neighbor classifiers that store more than one prototype. Since the computational expense of sampling increases linearly with the number of samples, we evaluate the accuracy of nearest neighbor classifiers that incorporate two prototypes per class in Table 3.5. There, we see a similar general pattern of prototype sampling accuracy as in Table 3.3. A significant improvement over a full nearest neighbor classifier is seen on three data sets (Diabetes, Iris Plants and Monks-2), poor results are seen on four data sets (Glass Recognition, Lymphography, Promoter and Soybean), and comparable results are seen on the remaining six data sets.

Table 3.5: Classification accuracy of prototype sampling (PS) algorithm, two prototypes per class.

Data	*NN*	*PS*	*Significance*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	72.5 ± 7.9	0.076
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	96.8 ± 2.0	0.247
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	78.7 ± 8.3	0.290
Diabetes $_{760}$	68.7 ± 5.1	75.0 ± 5.4	0.006
Glass Recognition $_{210}$	73.3 ± 10.3	63.3 ± 11.9	0.998
Hepatitis $_{150}$	77.3 ± 11.4	83.3 ± 10.5	0.073
Iris Plants $_{150}$	91.3 ± 7.1	96.7 ± 3.5	0.005
LED-24 Digit $_{200}$	36.5 ± 9.4	29.0 ± 8.1	0.905
LED-7 Digit $_{500}$	71.2 ± 7.7	72.2 ± 7.6	0.275
Lymphography $_{140}$	80.7 ± 8.9	72.9 ± 10.0	0.967
Monks-2 $_{430}$	48.8 ± 4.7	57.7 ± 5.5	0.002
Promoter $_{100}$	79.0 ± 9.9	70.0 ± 9.4	0.985
Soybean $_{300}$	91.0 ± 3.9	75.0 ± 7.4	1.000

Investigating the combination of classifiers that store varying numbers of prototypes may be an additional way to increase the diversity of a set of nearest neighbor classifiers. While this topic of clearly worthy of close investigation, for the reasons given, we have limited this investigation to a single prototype per class. In the remainder of this section, we examine some of the other factors that influence the accuracy of prototype sampling.

3.4.2.1 Number of Samples

If prototype sampling requires a large number of samples to achieve high accuracy, then sampling may be very inefficient. Of course, we are sampling a very large space. If there are s classes, $S_i, i = 1, \ldots, s$, then the number of prototype sets in the population, where one instance is taken from each class, is $\prod_{i=1}^{s} |T_{S_i}|$, where T_{S_i} is the set of training instances of class S_i. For example, with 90% of the 500 instances for the LED-7 task used for training, and ten classes, there are approximately $450^{10} \approx 3.4 \times 10^{26}$ minimal sets of prototypes in the population.

In this section, we investigate the effect of the number of samples on the generalization accuracy of the prototype set that best classifies the training set. We show that, for many data sets, a small number of samples yields accuracy as high as a much larger number. Since our approach to finding prototypes is driven by training set accuracy — but our real concern is test set accuracy — we would like to know when additional samples are not likely to improve test set accuracy.

In this experiment, we ran the prototype selection algorithm and traced its progress in the following way. For each data set we took 1000 prototype set samples, one prototype from each class. The prototypes were stored as the reference instances in a nearest neighbor classifier. For each data set, the nearest neighbor classifier was applied to the training set and to the test set. Each time the accuracy on the training set improved over the highest training accuracy previously achieved, we noted the corresponding test set accuracy. We then repeated this experiment 100 times, each time permuting the order of the prototype set samples. Since the samples are independent, this procedure allowed us to create alternative sets of samples by varying the order of the original samples. The training set consisted of 80% of the data set instances and the test set, 20%.

Example. An example may clarify the issues and the methodology. In Table 3.6, we give an extract from a trace of the prototype sampling algorithm on one ply of the Breast Cancer Ljubljana data set. The first column gives the sample number. (Since Common Lisp arrays begin with index 0, we begin sample numbers with 0.) The second column gives the training set accuracy of that sample; the test set accuracy is in the third column. Listed are all the samples for which training set accuracy was higher than its maximum up to that point. In samples that are not listed, the training accuracy was less than the then current maximal training accuracy. (The test accuracy may have been higher or lower than the previous highest test accuracy, of course, but these would not have been identified by the prototype selection algorithm. We track test accuracy, even though it would not be available to the prototype selection algorithm, in order to understand how the algorithm works.)

Table 3.6: Partial trace of prototype sampling algorithm on Breast Cancer Ljubljana data. Accuracies in percent correct.

Sample Number	*Training Accuracy*	*Test Accuracy*
0	43.42	53.45
1	61.40	77.59
2	64.47	75.86
3	67.54	70.69
16	67.98	75.86
21	68.86	75.86
23	72.37	79.31
72	72.81	75.86
385	73.68	79.31

Table 3.6 reveals that for only two samples out of 1000 (samples 1 and 23), did the training *and* test set accuracy (strictly) increase simultaneously over all of the samples previously seen. In other samples, either (a) the training set accuracy increased, but the corresponding test accuracy decreased, or (b) the training set accuracy did not increase over its current high.

For an example of behavior (a), the new maximal training set accuracy found in sample 72 actually led to a lower test set accuracy (75.86% vs. 79.31%). Sample 72 would have been adopted by the prototype selection algorithm as the then current best sample, since it displayed the highest training set accuracy up to that point. Unfortunately, this sample decreased the test set accuracy. Finally, in sample 385, a still higher training set accuracy was found (73.68%), but it led to the old high test set accuracy (79.31%) gained in sample 23. The post-hoc analysis of this example shows that the optimal stopping place for this run would have been at sample 23, before even 2.5% of the samples were taken.

Learning Curves. We would like to know the mean test set accuracy that is achieved after a given number of samples. Tracking this statistic will show how quickly on average the prototype sampling algorithm locates a set of prototypes that displays high generalization accuracy. Since these learning curves reflect test set accuracy, they are not available to the prototype sampling algorithm. The curves are useful only for analysis and explanation of performance after the fact. Figures 3.4, 3.5, 3.6 and 3.7 show graphs of the mean test set accuracy achieved by the prototype selection algorithm after i samples, $i = 1, \ldots, 1000$. One hundred repetitions were performed, as described above.

One clustering groups these graphs into five patterns.

Breast Cancer Ljubljana, Breast Cancer Wisconsin, Iris, LED-7. These graphs are characterized by a smooth increase in test accuracy and then an apparent leveling-off. Relative to a full nearest neighbor classifier, the sampling algorithm did comparatively well for data sets in this category (Table 3.3.)

Cleveland Heart Disease, Diabetes, Hepatitis. These charts show a quick increase in training accuracy and then a gradual or eventual slight decrease. Prototype sampling performed relatively well on these data sets.

LED-24, Lymphography, Monks-2, Soybean. The curves showed rapid increase in test accuracy, often to a sharp peak and then a dramatic decrease. Prototype sampling worked poorly on all these data sets.

Glass Recognition. Glass Recognition exhibited a slow increase in test accuracy and then an apparent leveling after a large number of samples. Prototype sampling worked poorly on this data set.

Promoter. Promoter showed a pattern of a rapid decrease followed by a gradual gain in test accuracy. Prototype sampling also worked poorly on this data set.

Mean sample accuracy for the data sets in the first two categories levels off after one hundred to several hundred samples. These are the data sets for which sampling performed well. It is

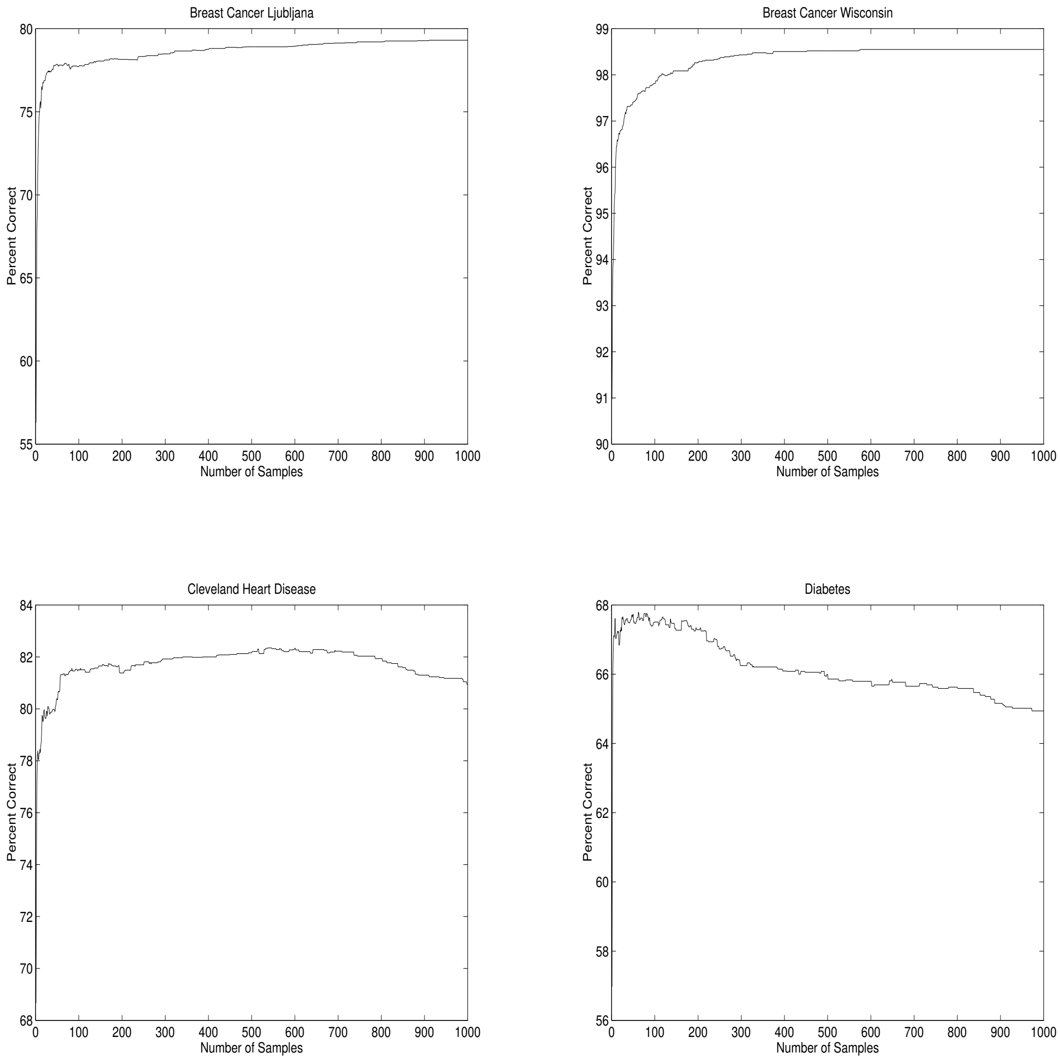

Figure 3.4: Relationship between test set accuracy ("Percent Correct") and number of samples taken.

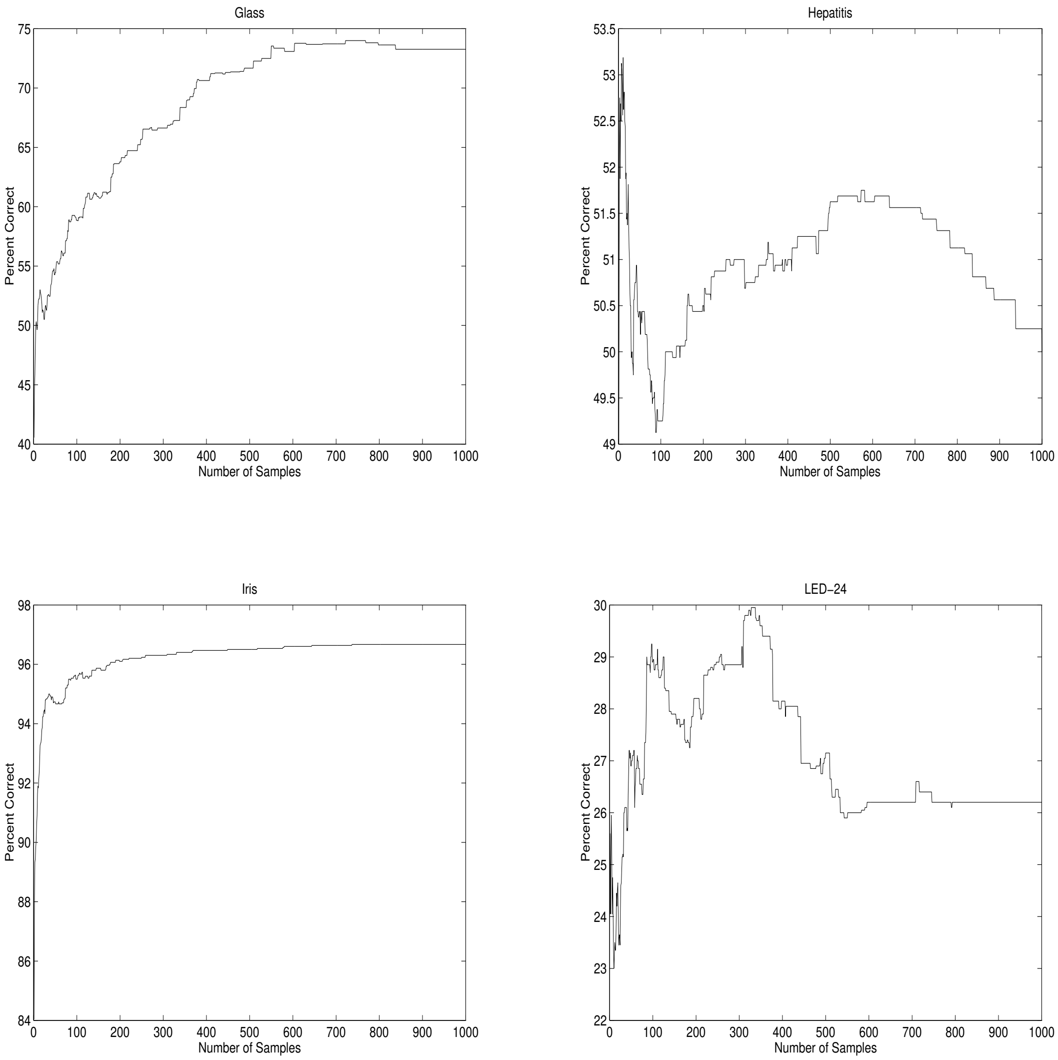

Figure 3.5: Relationship between test set accuracy ("Percent Correct") and number of samples taken.

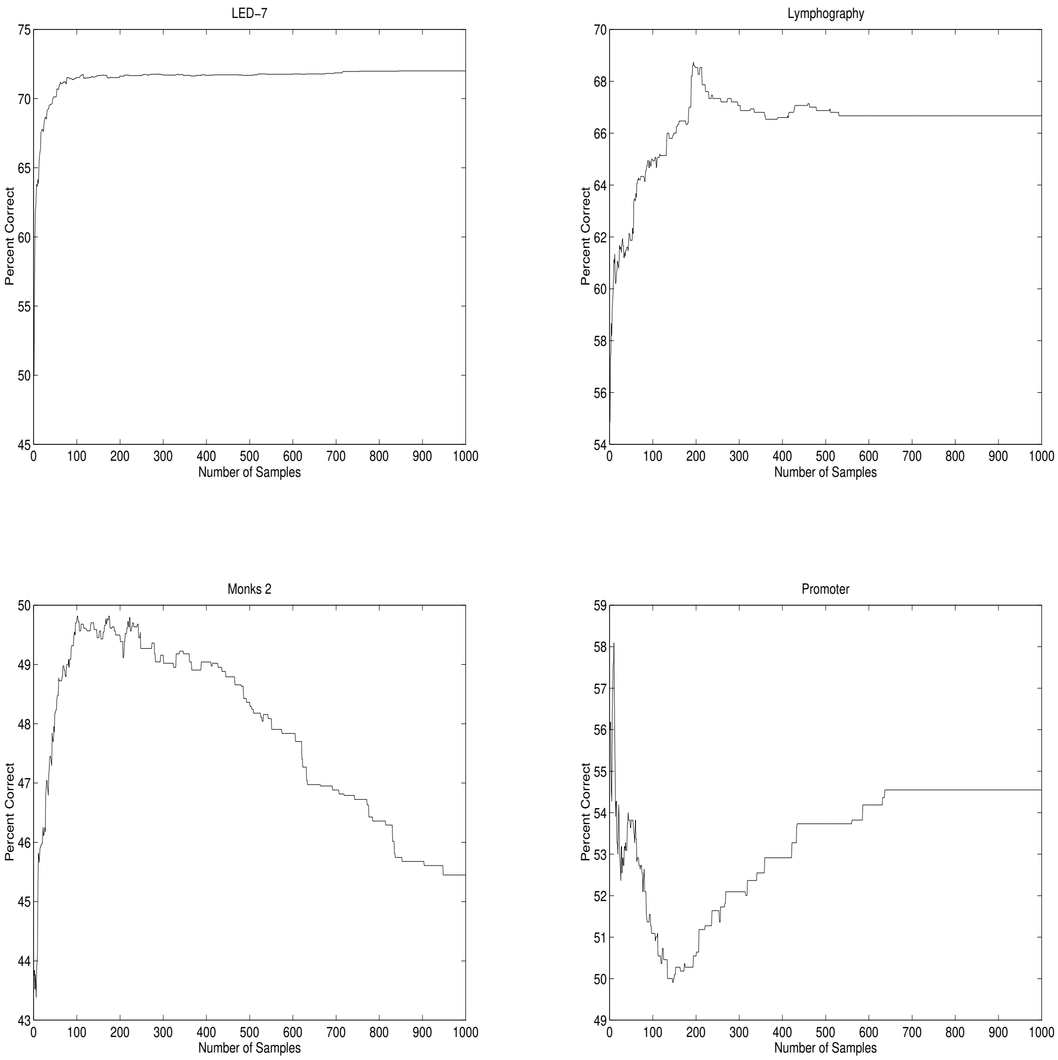

Figure 3.6: Relationship between test set accuracy (“Percent Correct”) and number of samples taken.

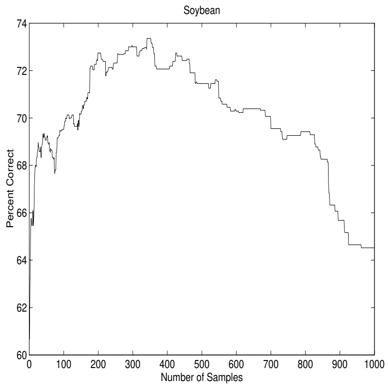

Figure 3.7: Relationship between test set accuracy (“Percent Correct”) and number of samples taken.

notable that for many of the 13 data sets, test accuracy decreases on average, or at best fails to increase, when more than several hundred samples are taken. The Glass Recognition data levels off after approximately 600 samples. Decreasing test accuracy is evidence of overfitting, suggesting that for some data sets, even these coarse classifiers can be overfit to the data. Maximal performance for most of the data sets is achieved after a fairly small number of samples.

Last simultaneous improvement in training and testing. These mean learning curves give one sense of how (on average) the test accuracy depends on the number of samples taken. Our analysis of Table 3.6 also hinted at another sense of this dependence. In that example, we found that highest numbered sample for which the training set and test set accuracy increased simultaneously over all the previous samples was sample 23, very early in the sampling of 1000 samples.

We ran the same experiment on the other 12 data sets and recorded the mean final sample number where training and test accuracy both increased over the corresponding values for the sample with the previous highest training accuracy. Table 3.7 gives the mean sample number (and standard deviation of sample number) of the sample where training set accuracy and test set accuracy last improved simultaneously. One thousand samples were taken and randomly permuted 100 times. Values shown are means taken over the 100 random permutations.

Table 3.7 suggests that the optimal stopping point for sampling comes after a fairly small number of samples: the mean is less than than 100. We had chosen 100 samples as our default much earlier in our research program, after experience with just four data sets [Skalak, 1994]. This retrospective analysis suggests that a higher number might have been reasonable for some data sets, such as Glass Recognition, but that 100 is a reasonable sample size for many of these data sets. This analysis provides retrospective support for our default sample plan, which selects 100 samples, although 200 samples may have performed better. The standard deviation of some of these sample sizes is large, however, which reflects a large variation in the number of the last sample where both training and test accuracy improved.

Table 3.7: Optimal number of samples to achieve maximal test accuracy.

Data Set	*Mean Sample Number*	*Std Dev.*
Breast Cancer Ljubljana $_{280}$	186	26
Breast Cancer Wisconsin $_{690}$	67	78
Cleveland Heart Disease $_{300}$	95	132
Diabetes $_{760}$	46	73
Glass Recognition $_{210}$	245	200
Hepatitis $_{150}$	54	86
Iris Plants $_{150}$	35	55
LED-24 Digit $_{200}$	107	161
LED-7 Digit $_{500}$	119	168
Lymphography $_{140}$	50	74
Monks-2 $_{430}$	54	81
Promoter $_{100}$	8	14
Soybean $_{300}$	152	178
Mean	93	
Median	67	

Comparison of Classification Accuracy. A final way to determine the effect of the number of samples is a direct comparison of test accuracies. Table 3.8 gives cross-validation test accuracies, comparing 100 samples vs. 1000 samples, with one prototype per class. On six data sets 100 samples does as well as or nominally better than 1000 samples; on seven data sets, 1000 samples does as well or nominally better. One thousand samples never significantly outperforms 100 samples at the 0.05 significance level. This experiment also shows that 1000 samples would have improved accuracy on Glass Recognition at a high level of confidence ($> 90\%$). The expense of 1000 samples is ten times as large, which becomes a more important consideration as more cases and features are present, and as more classifiers are trained.

These results provide evidence that in applying the prototype selection algorithm, a small number of samples (e.g., fewer than 100) often yields a test accuracy as high as a much larger number of samples (*viz.,* 1000). However, we do not know the effect of an even larger number of samples upon accuracy. Even if greater accuracy were obtained, the increase in accuracy would have to be traded against the substantial increase in the running time of the algorithm. Finally, maximizing the accuracy of one of these classifiers is not our only or ultimate goal. These experiments fall within the context of finding inexpensive classifiers that predict "reasonably"

Table 3.8: Comparison of test accuracies based on 100 samples and 1000 samples.

Data	*100 Samples*	*1000 Samples*	*Significance*
Breast Cancer Ljubljana $_{280}$	72.9 ± 8.9	72.1 ± 6.0	0.653
Breast Cancer Wisconsin $_{690}$	96.5 ± 2.5	96.4 ± 2.7	0.661
Cleveland Heart Disease $_{300}$	79.3 ± 7.2	80.3 ± 8.5	0.308
Diabetes $_{760}$	74.2 ± 6.2	74.9 ± 4.8	0.327
Glass Recognition $_{210}$	58.6 ± 9.0	62.9 ± 14.3	0.073
Hepatitis $_{150}$	82.0 ± 9.4	80.7 ± 11.5	0.637
Iris Plants $_{150}$	94.7 ± 5.3	94.7 ± 6.1	0.500
LED-24 Digit $_{200}$	29.0 ± 9.1	34.0 ± 10.2	0.140
LED-7 Digit $_{500}$	73.8 ± 9.4	74.8 ± 8.3	0.252
Lymphography $_{140}$	77.1 ± 8.1	72.1 ± 11.9	0.864
Monks-2 $_{430}$	43.3 ± 5.9	45.6 ± 6.1	0.149
Promoter $_{100}$	67.0 ± 12.5	66.0 ± 14.3	0.559
Soybean $_{300}$	64.0 ± 9.3	67.3 ± 8.4	0.190

well, so that we can use them as component classifiers to boost their accuracy or the accuracy of a full nearest neighbor classifier.

3.4.2.2 Clustering

Straightforward prototype sampling techniques do not always work, although their limits have not been determined theoretically or empirically. We have shown that the success of the sampling algorithm depends on the number of samples taken. In this section, we show that the success of a sampling algorithm depends partly on the distribution of instances in the data set and the geometric structure of the concepts to be learned.

In particular, one explanation for successful results from sampling is that some of the data sets exhibit well-defined, widely spaced classes in feature space, classes that exhibit a high degree of "internal coherence" and "external isolation." The intuition is that well separated and clustered classes moots the selection of a prototype, since any instance in an isolated class may be used as a prototype and still give perfect accuracy for that class. In this situation, a minimal 1-nearest neighbor classifier can perform very well.

Characterizing clusters and determining the "optimal" number of clusters in a real-valued data set are classical problems, and many indicators have been proposed, usually under the heading *stopping rules*, used to determine when to terminate a hierarchical clustering algorithm [Milligan and Cooper, 1985]. In an empirical examination of 30 stopping rules by Milligan and Cooper, the Calinski-Harabasz Index was the best performing procedure [Calinski and Harabasz, 1974; Milligan and Cooper, 1985]. The Calinski-Harabasz Index, I, is defined as

$$I = \frac{(trace\, B)/(k-1)}{(trace\, W)/(n-k)}$$

where n is the total number of data instances and k is the number of clusters in a clustering of the data. The *trace* of a square matrix is the sum of its diagonal elements. As defined in Appendix C, B is the between-cluster sum of squares and cross product matrix, and W is the pooled within-cluster sum of squares and cross product matrix from multivariate statistics [Johnson and Wichern, 1992]. One intuition behind this measure of clustering is that B captures the relative isolation of the classes and W captures their internal variability. As the isolation of the clusters grows larger and the internal variability grows smaller, the Calinski-Harabasz Index increases, indicating a higher degree of clustering.

We apply the Calinski-Harabasz Index not to determine when to terminate a hierarchical clustering algorithm, but to treat the classes themselves as clusters and measure the extent to which they are clustered or separated. We have applied this stopping rule to each data set that has only numeric features, since the Calinski-Harabasz Index does not handle symbolic features.

We performed an experiment to determine the effect of class isolation and cohesion, measured by the Calinski-Harabasz Index, on the test accuracy of the prototype sampling algorithm. Our hypothesis was that as the Calinski-Harabasz Index increased, indicating greater class cohesion and external isolation, sampling's performance would also increase.

In Table 3.9 we measure three quantities: (i) the correlation between the degree of clustering of the training set and the degree of clustering of the test set, both as measured by the Calinski-Harabasz Index; (ii) the statistical correlation between the degree of clustering on the *training* set and the mean accuracy on the *test* set; and (iii) the correlation between the degree of clustering on the *test* set and the mean accuracy on the *test* set. One hundred samples of prototype sets were taken, with one prototype per class. Ten-fold cross-validation was used.

Recall that the statistical correlation between two random variables X and Y, ρ_{XY}, is defined as

$$\rho_{XY} = E(\frac{X - \mu_X}{\sigma_X})(\frac{Y - \mu_Y}{\sigma_Y})$$

where E denotes expected value, μ denotes the arithmetic mean of a random variable, and σ denotes its standard deviation [Guttman *et al.*, 1971].

Table 3.9: Statistical correlation between (i) clustering of the training set and clustering of the test set, (ii) clustering of the training set and test accuracy, and (iii) clustering of the test set and test accuracy.

Data	*Cluster-Train Cluster-Test*	*Cluster-Train Test Accuracy*	*Cluster-Test Test Accuracy*
Breast Cancer Wisconsin $_{690}$	-0.68	0.07	0.21
Diabetes $_{760}$	-0.97	-0.81	0.87
Glass Recognition $_{210}$	-0.51	-0.72	0.44
Iris Plants $_{150}$	-0.75	-0.39	0.06
LED-24 Digit $_{200}$	-0.34	-0.58	-0.06
LED-7 Digit $_{500}$	-0.97	-0.57	0.52

This experiment reveals several regularities that may not have been anticipated. First, the correlation between the clustering of the training set and the test set is negative. This correlation means that as the training set is more highly clustered according to the Calinski-Harabasz Index, the test points are less well clustered, and vice-versa. The data and the Calinski-Harabasz Index behave as though the data set has a fixed "quantum of clustering", which is apportioned to the training and test sets. As the training set data are better clustered, the remaining points, held out for testing, are less well clustered.

Second, for all but the Breast Cancer Wisconsin data set, the degree of clustering of the classes on the training set is negatively correlated with the test set accuracy of the minimal nearest neighbor classifiers. One explanation is that, where the training data are more closely clustered, each of a number of prototypes from the cluster will serve equally well to classify the training data. In this situation, accuracy on the training set, used to select the prototypes, does not provide adequate reinforcement to distinguish class prototypes that will be accurate on test data from those that will not.

Our hypothesis did anticipate the third result. For all but LED-24 Digit data, the degree of clustering of the test set is positively correlated with higher test set accuracy. (The correlation for Iris is very low, however.) As the data to be classified are better clustered, the accuracy of a classifier is likely to increase, even where it stores a only small set of prototypes. The many noisy and irrelevant features of the LED-24 digit recognition task make a very difficult problem for a standard nearest neighbor classifier and for a minimal nearest neighbor classifier (as shown by the low accuracies in Table 3.3).

3.4.3 A Taxonomy of Instance Types

We have examined several factors that influence the success or failure of prototype sampling for creating minimal nearest neighbor classifiers. The algorithm is simple; it merely tries to find a prototype set that maximizes training accuracy. One reason that the sampling algorithm may not work on some data sets is that training set accuracy may not always be a good indication of test set accuracy. In turn, one explanation of a training-testing mismatch is that there are instances in the training set that are not good indicators of test accuracy. If there are training instances whose correct classification leads to poor test accuracy, then maximizing accuracy on the training set is not advisable. For example, if an instance has an erroneous class label, then probably it would not increase accuracy to classify correctly that mis-labeled instance. In this situation, removing such erroneously-labeled instances from the training set may leave a proper subset of the training instances that supports a faster and more robust fitness function than computing the accuracy on the entire training set.

Let us call the set of instances to which a sampled nearest neighbor classifier is applied to gauge its fitness the *prototype validation set.* By default, our prototype validation set is the training set. Here our goal is to use a subset of the training set as the prototype validation set, without decreasing accuracy. We show, in fact, that all our data sets contain instances that have the property that minimal nearest neighbor classifiers that classify those instances correctly will display lower, not higher, mean test accuracy.

In this section, we pursue the idea that maximal test accuracy may not be attained by correctly classifying certain training instances, and we suggest a taxonomy of instance types that arises pursuant to this notion. This taxonomy generalizes the idea of an *outlier*, an instance that has a strong influence on the calculation of some statistic, in particular a data instance that has been mis-entered or is not a member of the intended sample population [Wonnacott and Wonnacott, 1972].

The taxonomy is proposed to accelerate prototype sampling testing without decreasing accuracy, and to understand better the structure of the set of instances. A general lesson of case-based reasoning is that cases are not fungible and that they can play different roles in reasoning [Rissland, 1977; Rissland, 1978a; Rissland, 1989].

To create the taxonomy, we once again sample classifiers. The taxonomy has a simple idea at its root. Understanding how an instance is differentially classified by a large set of classifiers may provide a clue to the structure of the space of instances. An instance may be a reliable predictor of a classifier's test accuracy if the classifiers that classify it correctly have *higher* mean classification accuracy on the training set than those that classify it incorrectly. An instance may be a suspect predictor of a classifier's test accuracy if the classifiers that classify it correctly have *lower* mean classification accuracy on the training set than those that classify it incorrectly. So for each instance, we create and compare two distributions of classifier accuracies: (i) the training set accuracies of classifiers that correctly classify that instance and (ii) the training set accuracies of classifiers that incorrectly classify that instance.

Table 3.10 suggests a taxonomy that is based on two criteria: the number of times that an instance is classified correctly by a sample of classifiers, and whether the classifiers that classify the instance correctly have higher training accuracy on average (%CorR, for "Percent correct if right") than those that misclassify the instance (%CorW, for "Percent Correct if wrong"). First, we characterize qualitatively the number of times an instance is classified correctly as

High, *Medium* or *Low*. In the experiment below, we give a precise meaning to this informal characterization.

Table 3.10: Taxonomy of instances.

Type	*Frequency Correct*	*%CorR ? %CorW*
Core	High	>
Penumbral	Medium	>
Hard	Low	>
Misleading	High	<
Atypical	Medium	<
Outlying	Low	<

It may be surprising that correctly classifying some instances can be correlated with lower overall classification accuracy. If Outlying instances are present, for example, it may not be advantageous to classify them correctly. Even if they are classified correctly during training, the average classification accuracy is likely to suffer. This effect may be a reflection of overfitting the training data, despite the application of classifiers that create coarse hypotheses only. However, we may want to try affirmatively to get Outlying instances *wrong*. A similar notion is applied in the Deliberate Misclassification algorithm that we propose in Chapter 4 for component classifier construction. Next, we give an example to demonstrate the presence of each type of instance in one of our data sets.

Example. To create the data in Table 3.11, 100 samples of nearest neighbor classifiers were selected by sampling sets of prototypes, one from each class, from the Soybean data set. For each instance, we computed the following statistics, which were derived over the entire sample of classifiers:

- the number of the sampled classifiers that correctly classify this instance (the *frequency correct*, in the "Freq.Cor." column)

- the mean accuracy over a training set of the classifiers that correctly classified this instance ("%CorR")

- the mean accuracy over a training set of the classifiers that incorrectly classified this instance ("%CorW")

- the statistical significance according to a t-test for a difference in the means of the previous two items ("Sig.")

We then culled an example from each class by hand to illustrate each taxonomic category.

Table 3.11: Examples of instances in the taxonomy from the Soybean data set.

Category	*Case*	*Freq.Cor.*	*%CorR*	*%CorW*	*Sig.*	*Class*
Core	260	89	53.95	46.90	0.000	frog-eye-leaf-spot
Penumbral	240	54	55.05	50.97	0.000	alternarial-leaf-spot
Hard	163	15	55.80	52.71	0.032	anthracnose
Misleading	2	90	52.84	56.20	0.052	diaporthe-stem-canker
Atypical	50	62	52.25	54.68	0.023	phytophthora-rot
Outlying	283	14	48.84	53.88	0.000	frog-eye-leaf-spot

Instance 260 This instance is often correct, and the classifiers that classify it correctly have a higher mean accuracy than those that get it wrong. This is a *core* instance.

Instance 240 This instance is right a median number of times (the median is 54), and the accuracy of those classifiers that get it right show higher accuracy than those that get it wrong. This is a *penumbral* instance.

Instance 163 This instance is rarely gotten right, but the classifiers that get it right have higher accuracy than those that get it wrong. This is a *hard* instance.

Instance 2 This instance is often gotten right, but the classifiers that get it right have lower accuracy than those that get it wrong. This instance is *misleading*, because it appears to be easy to get right, but leads to lower accuracy classifiers on average.

Instance 50 This instance is right about a median number of times, but the classifiers that get it right have lower accuracy than those that get it wrong. This instance is *atypical*.

Instance 283 This instance is rarely gotten right: it's difficult. In addition, classifiers that get it right actually have lower mean accuracy than those that get it wrong. This instance is *outlying*.

A next logical question is how many instances of each type appear in the Soybean and other data sets; answers are given in Table 3.12. In that table, "Undiff." means "undifferentiated," which is the leftover category for instances that do not fall into any of the other categories. An instance is placed into the Core, Penumbral or Hard categories if the mean training accuracy of classifiers that classify it correctly is significantly better (at the 95% confidence level) than the accuracy of those classifiers that classify it incorrectly. If this condition is satisfied, it was placed into one of those three categories according to the frequency correct, the number of times instance was correct classified by the classifiers in the sample. If the frequency correct for an instance was within (plus or minus) one standard deviation of the mean frequency, it was put in the Penumbral category. If the frequency correct was more than one standard deviation above the mean, it was placed into the Core category; less than one standard deviation, the Hard category. Analogous definitions were applied to the instances for which the classifiers that misclassified the instance had higher mean classification accuracy than those the correctly classified it, which were placed in the Misleading, Atypical and Outlying categories.

Misleading instances — instances that are often correctly classified but that lead to lower mean test accuracy — are rare. If a data set is fairly easy, with mean accuracy for a minimal nearest neighbor classifier above 90% (e.g., Breast Cancer Wisconsin and Iris), there may be no Core instances. If nearly all the classifiers in a sample correctly classify an instance (say, 98 or 99, out of 100), there may be no classifier whose frequency correct is more than one standard deviation above the mean frequency correct.

A coarser taxonomy might group Core, Penumbral and Hard instances under the heading "Characteristic", and Misleading, Atypical and Outlying instances under the heading "Uncharacteristic".

Table 3.12: Average number of instances of each type in the taxonomy.

Data	*Undiff.*	*Core*	*Pen.*	*Hard*	*Mis.*	*Atyp.*	*Outly.*
Breast Cancer Ljubljana $_{280}$	35.0	33.4	140.4	1.4	0.0	10.4	37.4
Breast Cancer Wisconsin $_{690}$	403.8	0.0	151.5	40.0	0.0	21.6	13.1
Cleveland Heart Disease $_{300}$	49.6	23.9	167.9	1.7	0.0	1.8	28.1
Diabetes $_{760}$	70.6	21.7	77.7	3.0	0.3	13.1	6.6
Glass Recognition $_{210}$	25.4	7.0	91.6	1.6	0.0	1.1	13.3
Hepatitis $_{150}$	52.0	0.0	63.0	3.9	5.1	6.2	4.8
Iris Plants $_{150}$	127.5	12.3	32.2	3.5	0.3	2.4	1.8
LED-24 Digit $_{200}$	144.0	78.0	198.8	3.4	0.0	3.5	22.3
LED-7 Digit $_{500}$	46.7	25.2	51.2	3.2	0.0	2.1	5.6
Lymphography $_{140}$	225.1	17.8	84.5	4.4	3.5	27.6	26.1
Monks-2 $_{430}$	151.4	123.6	282.0	6.0	0.0	39.2	89.8
Promoter $_{100}$	56.1	8.1	29.2	2.1	0.0	0.1	0.4
Soybean $_{300}$	153.4	15.4	79.8	8.5	1.7	5.1	13.1

This taxonomy now suggests a way to limit the set of instances that are in the prototype validation set, the instances to which a set of prototypes is applied in order to judge its accuracy. As we observed, normally, we use the training set itself for validation. On the other hand, if the Uncharacteristic instances are removed from the prototype validation set, then test accuracy may be improved. Also, once these Uncharacteristic instances are removed, the evaluation of each set of prototypes sampled will be accelerated, since there will be fewer instances to classify.

Table 3.13 shows that for the prototype selection algorithm, accuracy is never significantly increased at the $p \leq 0.05$ level by leaving the Misleading, Atypical and Outlying (Uncharacteristic) instances in the validation set. Thus, there is a savings in the number of instances tested in the prototype validation set. (The numbers of Uncharacteristic instances, which would be removed from the validation set, are in the last three columns of Table 3.12). On eight data sets, removing these instances from the validation set yields equal or nominally higher accuracy at lower computational expense. For some data sets (Promoter, for example), however, there are few Uncharacteristic instances and the savings in runtime would be small.

This approach will only be useful where a large number of samples are to be taken *after* the prototype validation set is pruned, in order to realize benefits greater than the initial costs of finding the Uncharacteristic instances. For example, where a very small number of instances

Table 3.13: Classification accuracy of the Prototype Sampling *(PS)* algorithm versus a variant of the Prototype Sampling algorithm *(PS-Char)* that leaves out Misleading, Atypical and Outlying instances from the prototype validation set.

Data	*PS-Char*	*PS*	*Sig.*
Breast Cancer Ljubljana $_{280}$	73.6 ± 8.9	72.9 ± 8.9	0.661
Breast Cancer Wisconsin $_{690}$	96.8 ± 1.6	96.5 ± 2.5	0.679
Cleveland Heart Disease $_{300}$	80.0 ± 5.7	79.3 ± 7.2	0.778
Diabetes $_{760}$	72.9 ± 7.3	74.2 ± 6.2	0.250
Glass Recognition $_{210}$	57.1 ± 13.7	58.6 ± 9.0	0.280
Hepatitis $_{150}$	82.7 ± 10.0	82.0 ± 9.4	0.610
Iris Plants $_{150}$	94.7 ± 5.3	94.7 ± 5.3	0.500
LED-24 Digit $_{200}$	30.5 ± 10.9	29.0 ± 9.1	0.646
LED-7 Digit $_{500}$	72.6 ± 6.5	73.8 ± 9.4	0.280
Lymphography $_{140}$	70.0 ± 13.0	77.1 ± 8.1	0.074
Monks-2 $_{430}$	47.0 ± 5.8	43.3 ± 5.9	0.961
Promoter $_{100}$	65.0 ± 9.7	67.0 ± 12.5	0.339
Soybean $_{300}$	65.7 ± 10.3	64.0 ± 9.3	0.793

are Uncharacteristic, as in the Promoter data set, this approach will not be worth the added expense of the preliminary sampling.

This general approach of studying the distributions of a sample of classifiers when applied to individual training instances is applicable to other types of classifiers, not just to nearest neighbor classifiers. It may work more efficiently with minimal nearest neighbor classifiers, because they are fast to train and apply. Training is non-existent and only comparisons with a few prototypes need be made to apply the classifier.

The results in this section show that instances in these data sets exhibit different behaviors when minimal nearest neighbor classifiers are applied to them. It is not necessarily a good idea to classify some training instances correctly. Identifying the Uncharacteristic instances enables the algorithm to accelerate the sampling procedure by validating a prototype set only on a proper subset of training instances. However, one drawback of this method is the cost associated with sampling to identify the Characteristic and Uncharacteristic instances. Nevertheless, this approach does reveal some structure in a space of instances as induced by a set of classifiers, regardless of its computational efficiency.

3.5 Summary

This chapter has been a bridge between composite classifier construction research to date and the approach applied in this thesis. The first span of the bridge was built by arguing that the design goals that underlie classifier combination research (component accuracy, diversity and efficiency) could be advanced by an approach characterized by a set of related, but untried, techniques (homogeneous nearest neighbor classifier components, prototype selection through sampling, coarse classifier hypotheses, and concise level-1 representation). Observing these constraints also allows us to propose solutions to a number of open questions posed in the research literature.

The bridge's second span was built by investigating one aspect of our approach, prototype sampling, which will be part of the combination algorithms we describe in the next two chapters. In this chapter, we evaluated prototype sampling for a preliminary task, creating an independent nearest neighbor classifier. Prototype sampling showed significantly increased accuracy on three data sets, but performed very poorly on four others. The influence of several factors on prototype sampling accuracy and efficiency was evaluated, including class clustering, the number of samples drawn, and the presence of uncharacteristic instances in these data sets. Investigating this last factor led to a taxonomy of instance types created by analyzing the distribution of classifier training accuracies as a sample of classifiers are applied to each instance.

With this bridge, in the next two chapters, we evaluate our proposed approach to classifier combination within two composite architectures — boosting and stacked generalization.

CHAPTER 4

BOOSTING

Boosting is the first combination task that we investigate. Informally, boosting is the task of improving the accuracy of a given classifier. More formally, the *hypothesis boosting problem* is to show that *weak learnability* implies *strong learnability* in the Probably Approximately Correct (PAC) theoretical model of learning [Schapire, 1990]. A class of concepts is strongly learnable in the PAC model if there is a polynomial-time algorithm that, with high confidence, can learn any concept in the class with a small error. A class of concepts is weakly learnable if, with high confidence, any concept in the class can be learned with an error that is slightly lower than random guessing. Schapire showed that these two theoretical notions of learnability are equivalent [1990]. In this chapter, we investigate the applied problem of boosting the accuracy of a given nearest neighbor classifier.

In this chapter we propose and evaluate four algorithms for nearest neighbor boosting, called *Coarse Reclassification*, *Deliberate Misclassification*, *Composite Fitness* and *Composite Fitness–Feature Selection.* Since the chapter is long, it may help to keep in mind the following structure. (The table of contents may be consulted for a reminder of the structure, also.)

Conceptually, the first half of the chapter is devoted to describing and evaluating the four algorithms individually (Sections 4.1 to 4.5). The first section introduces each algorithm and explains why it is included in this study (Section 4.1). Readers interested only in a general description of the algorithms may want to read that section and skim the rest of the chapter. For readers interested in detail, descriptions of each algorithm are given in the next four sections (Section 4.2 to Section 4.5). Each algorithm is evaluated on the task of boosting a nearest neighbor classifier, to answer the question, *How well does each of these algorithms boost?* On five of the 13 data sets in this collection, a significant improvement ($p \leq 0.05$) over a nearest neighbor algorithm is shown by at least one of these algorithms, which add only one, minimal component classifier for boosting.

The second half of this chapter (Sections 4.6, 4.7 and 4.8) is devoted to comparing the four algorithms to each other and to a classic boosting algorithm [Schapire, 1990].

Our specific objective in the inter-algorithm comparison (Section 4.6) is to answer the inquiries, *How well do these algorithms boost, when compared to each other?* and *How do these algorithms perform boosting?*. The answers we give are that these algorithms exhibit similar composite accuracy for many of these data sets (the *how well*), but that the *manner* in which they boost is different, for the most part (the *how*). The manner in which they boost is characterized in terms of the accuracy and diversity of the component classifiers created by each algorithm.

To answer the question, *How well do these algorithms work, relative to another boosting algorithm?*, we compare the four algorithms to an implementation of Schapire's original boosting algorithm [Schapire, 1990] (Section 4.7). On the task we study — boosting a full nearest neighbor classifier using minimal component classifiers — we show that on an average of 3.5 data sets of the 13, the four algorithms produce composite classifiers that are significantly more accurate than those created by that boosting method.

The final section (Section 4.8) is devoted to the summary questions, *When do the four boosting algorithms fail?* (Section 4.8.1) and *When do they succeed?* (Section 4.8.2). We show that the four algorithms tend to do poorly on data sets that have a small number of instances per class. As to their success, we observe that the boosting algorithms are selectively superior [Brodley, 1994], and that each algorithm is superior to the others on at least one of the data sets in this collection.

The established terminology of boosting may be somewhat confusing. If we are given a specific classifier that has been trained on some set of training instances, boosting the accuracy of the classifier does not increase *that classifier's* accuracy. Rather, in boosting research to date, a more accurate classifier is created by combining the predictions of the given classifier with those of other classifiers. There may be other — yet undiscovered — approaches to boosting, of course. The boosting terminology is analogous to the commonplace notion of "getting a boost." If you want to see over a fence, you ask a friend to lift you — to give you a boost. By combining your efforts with your friend's, your perspective is improved.

4.1 Four Algorithms for Boosting

In this section we describe the four boosting algorithms, provide the intuition behind each of them, and show their connection to other research. The algorithms are described and evaluated in detail in separate sections following this introductory section.

Our approach to boosting is to incorporate the given (*base*) classifier as a level-0 classifier in a stacked generalizer whose level-1 learning algorithm is ID3 [Quinlan, 1986]. We postpone the justification of the choice of ID3 until Chapter 5, where we investigate three level-1 combining algorithms: ID3, nearest neighbor and voting. There, we show that ID3 and nearest neighbor have comparable accuracy on the data sets in our collection, and that both outperform voting where the components are inaccurate.

Our objective is to construct one other (*complementary*) level-0 classifier, so that a stacked generalizer that uses the base classifier and the complementary classifier as level-0 components has higher generalization accuracy than the base classifier (Figure 4.1). The complementary classifier we construct will be a minimal nearest neighbor classifier, which stores only one prototype per class.

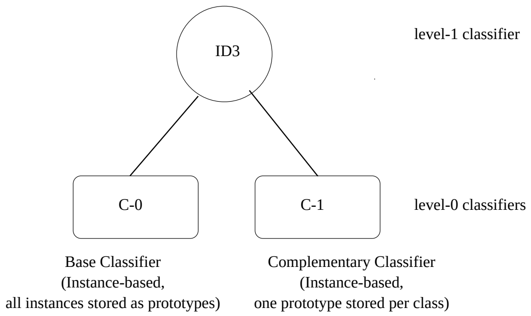

Figure 4.1: Boosting architecture for this chapter.

Common to these four algorithms is a basic structure: each samples minimal nearest neighbor classifiers and maximizes a fitness function to select a minimal classifier to use as a level-0 component together with a full nearest neighbor classifier. Coarse Reclassification and Deliberate Misclassification use the accuracy of the sampled *component* to direct the search for a complementary classifier. Composite Fitness and Composite Fitness–Feature Selection use the accuracy of the entire *composite* classifier to guide the search for a complementary classifier (Table 4.1).

There are additional relationships between pairs of the algorithms. Deliberate Misclassification is a variation of Coarse Reclassification; Composite Fitness–Feature Selection is a variation of Composite Fitness. Each variation attempts to create component classifiers that are more diverse than those added by its sibling algorithm, to determine if an additional measure of diversity yields better boosting. Recall that diversity is a measure of the extent to which the components make different errors (Chapter 2). We shall define the diversity of two classifiers as the fraction of the set of instances that are classified incorrectly by one classifier but not by the other. This definition is a specialization of the diversity measure applied by Ali and Pazzani [1996], described in Section 2.1.

In brief, Coarse Reclassification selects the most accurate sampled component. Deliberate Misclassification is a variation that attempts to increase component diversity, by selecting a component that is the most accurate on a *modification* of the original training set. Composite Fitness selects a sampled complementary component classifier whose fitness is determined by the accuracy of the resulting composite classifier that incorporates that component. Composite Fitness–Feature Selection is a variation of the Composite Fitness algorithm, which severely limits the number of features considered in the distance metric by the complementary component nearest neighbor classifier.

4.1.1 Coarse Reclassification

Component accuracy has been observed to be an important factor in the success of stacked generalizers, possibly the most important [Ali and Pazzani, 1996; Opitz and Shavlik, 1995]. In this first algorithm, Coarse Reclassification (CR), we present a straightforward boosting algorithm that tries to maximize the accuracy of a complementary component classifier. We

Table 4.1: Boosting algorithms summary.

Fitness Function	**Component Training Accuracy**	**Composite Training Accuracy**
Increase Training Accuracy	Coarse Reclassification	Composite Fitness
Increase Training Accuracy and Component Diversity	Deliberate Misclassification	Composite Fitness–Feature Selection

show that this simple algorithm boosts nearest neighbor accuracy at statistically significant levels on a variety of the data sets tested.

Coarse Reclassification uses the same idea as Breiman's bagging [1994a]: the utility of resampling a training set to create classifiers that are to be combined. However, bagging nearest neighbor classifiers reportedly did not work in Breiman's experiments because bagging's bootstrap sampling of the training set yielded stable nearest neighbor classifier components [1994a]. Breiman does not formally define stability, but uses the term to refer to the condition that leaving a small number training instances out of the training set (or putting a few more in) tends not to change the classifier concept hypothesis very much. For nearest neighbor classifiers, which store the training instances as prototypes, stability entails little change in the concept hypothesis when a few prototypes are left out. Bagging relies on bootstrap sampling, which resamples the training set with replacement to yield as many points as there are training instances. Typically in a bootstrap sample, a number of instances are repeated and a number are left out. Instances drawn more than once from the training set do not affect a 1-nearest neighbor classifier, but might affect other learning algorithms.

To overcome the problem of nearest neighbor classifier stability, the Coarse Reclassification algorithm (and the others we investigate) resamples the training set, but draws only a small number of prototypes in each sample. Storing few prototypes yields an unstable classifier, in the sense that substituting one prototype for another can make a large difference in the concept hypothesis of the classifier. The intuition behind Coarse Reclassification is that even though, in general, classifier instability is probably better avoided, ironically, the deliberate introduction of instability may yield classifiers that may be combined more effectively. *Radical destabilization* is the name we give to this policy of deliberately introducing a large measure of instability.

4.1.2 Deliberate Misclassification

Deliberate Misclassification (DM) is designed to improve Coarse Reclassification, by increasing the diversity of the level-0 components.

Before proceeding further, let us define formally the diversity of two classifiers, giving the definition that we apply in this and succeeding chapters. Let C_0 and C_1 be classifiers; T, a set of instances in the domain of C_0 and C_1; and $E(C_i, T)$, the set of errors that C_i makes when C_i is applied to T, $i = 0, 1$. For notational convenience, we drop T as a functional argument, and assume that it is fixed throughout each discussion. We define the *diversity* of C_0 and C_1, $div(C_0, C_1)$, as the fraction of the set T that is in the symmetric difference of $E(C_0)$ and $E(C_1)$:

$$div(C_0, C_1) = \frac{|E(C_0) \bigtriangleup E(C_1)|}{|T|} = \frac{|(E(C_0) - E(C_1)) \cup (E(C_1) - E(C_0))|}{|T|}.$$

As surveyed in Chapter 2, research to date has applied at least seven different approaches to creating component diversity. We try to draw a lesson from one of the more successful classifier combination algorithms: Error Correction Output Coding (ECOC), an algorithm that uses stacked generalization for multi-class learning problems [Dietterich and Bakiri, 1995], discussed in Section 2.2.1.2. ECOC is not a boosting algorithm, however, and it is not obvious how it may be adapted for boosting. ECOC trains a set of component classifiers on a set of two-class problems that are derived from the original multi-class problem. The objective is to use the component classifiers to create a level-1 representation that constitutes an error-correcting binary code. An error-correcting binary code is designed to provide robustness: it permits some of the component classifiers to make errors without affecting composite accuracy.

Using C4.5 trees as component classifiers, ECOC was more accurate than a baseline C4.5 tree on six of eight data sets, at the cost of combining a large number of component classifiers. An analysis of ECOC showed that one of its sources of power is that the inter-class boundary between every pair of classes is learned many times [Kong and Dietterich, 1995]. However, it may be unnecessary to re-learn every boundary, since some boundaries may not be the source of errors. More component diversity may also be introduced by forcing components to learn slightly *different* inter-class boundaries, rather than the same ones time and again.

The tack we take in Deliberate Misclassification draws from the simple observation that the task is supervised learning: each instance has an associated class label that we try to predict. Class labels are not altered by any of the seven diversification techniques. Generally speaking, these techniques change the model class, introduce randomization or resample the instances as presented, but none uses our inherent power to modify class labels in training a classifier. By changing some instance class labels in the training phase, we may increase diversity. Modifying training instances in this way would force a minimal nearest neighbor classifier to learn slightly different inter-class boundaries from the base classifier, rather than the same ones a number of times, as done by ECOC. We also change labels only along boundaries where errors have been committed by the base classifier, so that not every boundary is re-learned.

Deliberate Misclassification also draws a lesson from a technique used by Schapire to decrease the time and space complexity of his boosting algorithm [1990]. For two-class problems, Schapire shows that stochastically modifying the hypothesis created by the first component classifier can reduce the time that the other two components spend waiting for examples to add to their training set. As we shall see, the re-targeting approach underlying Deliberate Misclassification differs from this boosting variant in several ways. The approach we propose (1) is not random, but deterministic; (2) is applicable to multi-class problems; and (3) is designed to increase the accuracy of the resulting composite classifier, not just to decrease the computational complexity of the algorithm.

We investigate the utility of modifying class labels by evaluating a heuristic algorithm for changing training instance class labels, the Deliberate Misclassification (DM) algorithm. First, the base classifier is applied to the training set and the errors are compiled. Based on these errors, the algorithm modifies the class labels of certain training instances. This modified training set is then used to train a complementary component classifier. The algorithm is so named because it deliberately (rather than randomly) tries to misclassify certain instances by training a component classifier on mis-labeled instances.

At this point, we are halfway through an introduction to the four boosting algorithms. We have discussed the two algorithms that use the training accuracy of a component classifier as the fitness function to guide the search for a set of prototypes. The next two algorithms use the training accuracy of the entire composite classifier to guide the search for prototypes: Composite Fitness and a variation, Composite Fitness–Feature Selection.

4.1.3 Composite Fitness

The intuition behind the Composite Fitness (CF) algorithm is that it makes good sense in training to attempt to maximize the accuracy of the entire composite classifier, rather than to maximize any criteria that apply solely to a component, such as accuracy or diversity. Little work has been done to investigate this approach, but the utility of simultaneously training the components and the combining classifier is a characteristic of the hierarchical mixture of experts algorithm [Jacobs, Jordan and Barto, 1991; Jordan and Jacobs, 1993]. See page 42. In that algorithm, simultaneous training of the various levels of a stacked generalizer created components that were effective on different inputs. In addition, the task decomposition was *learned* by the composite classifier, rather than assigned to the components *a priori*. Here, we do not aim to partition the input space or to divide tasks between the components, but to find components whose predictions can be combined effectively for instances throughout the input space. There are several reasons why maximizing composite training accuracy is worth investigation as an approach to select component classifiers.

First, maximizing training accuracy of the composite classifier makes sense because it is composite accuracy that we want to maximize — albeit composite accuracy on the test set. In this sense, maximizing composite accuracy may provide a more direct approach than one that maximizes component accuracy.

The second reason for appealing to composite accuracy is that it is unclear what fitness function of the component classifiers should be maximized. In addition to component accuracy and diversity, other factors may be relevant that have not yet been identified. Unless these factors are built explicitly into a fitness function, it would be hard to incorporate them. Maximizing composite accuracy attempts to incorporate implicitly the influence of all factors, both known and unknown.

An alternative approach, taken by Opitz and Shavlik, is to assume that a linear combination of accuracy and diversity should be maximized, and they use a genetic algorithm to search for components that maximize a linear function of accuracy and diversity [1995]. By contrast, the Composite Fitness algorithm allows the contributions of these two factors to be determined implicitly, whatever their relationship.

One difficulty in using component accuracy and diversity to maximize composite test accuracy is illustrated by the graph in Figure 4.2. In this experiment, we took a random training-test split of the Soybean data (90%-10%) and trained 100 stacked generalizers on the training set. Each stacked generalizer used two level-0 component classifiers, a full nearest neighbor classifier and a minimal one, and ID3 as the level-1 learning algorithm. The prototypes for the minimal classifier were drawn at random, one from each class. This is the architecture for our boosting experiments.

Figure 4.2 depicts the stacked generalizer test accuracy graphed against component accuracy and diversity on the training set. This contour graph is a post-hoc illustration of how composite test accuracy depends on two factors computed and available during training. Diversity was defined as usual, here with respect to the training set, as the portion of the training set for which the two classifiers disagreed but where one of them was correct. An algorithm that appealed directly to component accuracy and diversity to maximize composite test accuracy would confront a landscape like that depicted in the figure. The landscape has many peaks (light regions) and valleys (dark regions), making for precarious hillclimbing. In fact, towards the center of the graph, points representing the highest composite test accuracy are close to those with the lowest accuracy. Discontinuities of this sort make it difficult to select composite classifiers with high test-set accuracy on the basis of component accuracy and diversity on the training set.

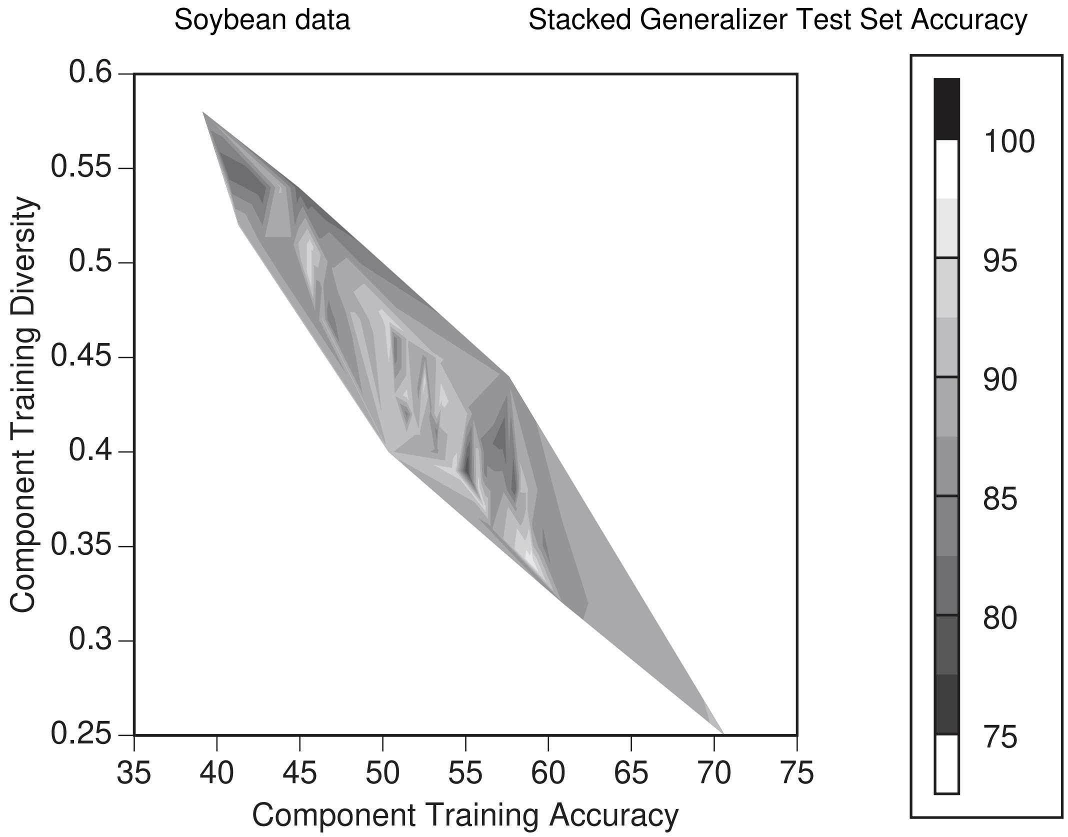

Figure 4.2: Contour graph of test accuracy of stacked generalizers on Soybean data. Stacked generalizer test accuracy (%) is graphed against the complementary component training accuracy (%) and the diversity of the components on the training set. (The legend format provided by the graphing package may be confusing; the top and bottom intervals of the legend are apparently provided for contrast.)

There is a third reason to apply a fitness function that uses composite accuracy to guide prototype search. By appealing to composite accuracy, the level-1 learning algorithm is thereby taken into consideration. A fitness function that looks only to the level-0 components does not take account of the learning bias of the level-1 algorithm. In particular, using composite accuracy as a fitness function helps to ensure that the diversity between the two component classifiers can be exploited by a level-1 combining algorithm. Diversity "for its own sake," diversity from which the level-1 learning algorithm cannot induce a generalization, will not lead to higher composite accuracy.

For these reasons, we have designed a nearest neighbor boosting algorithm that works as follows. The Composite Fitness algorithm repeatedly samples a set of prototypes to be stored in the complementary component classifier; in turn, incorporates each complementary component into a composite classifier; trains the resulting composite classifier on the training set; and then computes the training accuracy of the composite classifier. The prototype set that yields the composite classifier that has the highest training accuracy is used for the complementary component. The algorithm outputs the composite classifier with the highest training accuracy.

In the final section of this introduction, we describe an algorithm that uses this same procedure, but also limits the features incorporated in the distance metric used by the complementary nearest neighbor classifier.

4.1.4 Composite Fitness–Feature Selection

The Composite Fitness–Feature Selection (CF-FS) algorithm is a variation of the Composite Fitness algorithm, in that it uses the training accuracy of the entire stacked generalizer to select a complementary classifier. The Feature Selection version of the algorithm is distinguished by the distance metric applied by the complementary classifier. The complementary nearest neighbor component classifier only takes one feature (*sic*) into consideration to compute a distance between two instances. This feature is selected by an exhaustive search of the individual features. The intuition behind selecting only one feature is that the fewer features that are used, the more diversity the complementary classifier is likely to provide.

Procedurally, for each prototype set sampled and for each feature, this algorithm creates a minimal nearest neighbor classifier that stores those prototypes and solely applies that individual feature in its distance metric. For example, if ten prototype sets are sampled and there are nine features, 90 complementary nearest neighbor classifiers are created. To determine the utility of each classifier, it is incorporated into a stacked generalizer as a complementary component classifier, and the resulting stacked generalizer is trained and applied to the training set. The stacked generalizer with the highest training accuracy is output by the Composite Fitness–Feature Selection algorithm.

This algorithm takes a lesson from the established importance of feature selection to accurate statistical pattern recognition and theoretical and applied inductive learning (*e.g.*, [Littlestone, 1987; Krzanowski, 1988; Weiss and Kapouleas, 1989; Tan and Schlimmer, 1990; Callan, 1993; Ripley, 1996]). Here, however, we use feature selection not as a technique to increase accuracy, but to increase diversity. This algorithm also draws from the NetTalk example of Wolpert [1992], which selected inputs to component classifiers by hand (Section 2.2.1.1), and from the work of Battiti and Colla [1994], who observed that networks that used distinct input representations (also selected by hand) showed the greatest improvement on a handwriting recognition problem. Also, Breiman created diverse linear classifiers by applying Stepwise Backward Elimination of variables [1992], a standard statistical feature selection algorithm that begins with all features and iteratively eliminates features that do not increase training accuracy [Krzanowski, 1988]. Our feature selection search, on the other hand, is the first step of the Forward Sequential Selection algorithm, which starts with an empty feature set and iteratively adds features that improve accuracy [Krzanowski, 1988]. We do not apply the Forward Sequential Selection algorithm past the first iteration because our goal is not to create the most accurate complementary component, but to create diverse components.

4.1.5 Summary

In this chapter, we propose four algorithms to boost nearest neighbor accuracy. These algorithms explore different means to create diverse and accurate component classifiers.

Table 4.2: Techniques used by four nearest neighbor boosting algorithms to create diverse component classifiers.

Algorithm	*Technique*
Coarse Reclassification	sampling minimal classifiers
Deliberate Misclassification	modification of training instance class targets
Composite Fitness	maximization of composite training accuracy
Composite Fitness–Feature Selection	feature selection

Two of these algorithms use the accuracy of the *component* classifier to select prototypes; two use the accuracy of the *composite* classifier to select prototypes and, optionally, to select a feature for a complementary classifier. These latter two algorithms attempt to account for accuracy and diversity implicitly. We shall conclude that a minimal nearest neighbor classifier can be combined with a full nearest neighbor classifier to boost its accuracy significantly on a variety of the data sets in this collection.

Our detailed description of each of the four boosting algorithms will follow the same form. We describe the algorithm and give pseudocode for it. We then present an experiment that evaluates its performance on our basic task: boosting a baseline nearest neighbor classifier. After all algorithms have been described and evaluated on the basic task, we compare them to Coarse Reclassification, which serves as a simple baseline boosting algorithm. We then analyze the performance of these algorithms on the data sets tested.

4.2 Coarse Reclassification

In this section we show that the Coarse Reclassification algorithm boosts nearest neighbor generalization accuracy on a variety of data sets by trying to maximize the training accuracy of the complementary component classifier.

First, we define notation that will be used in the rest of the dissertation. A set of training instances will be denoted by $T = \{x_j | j = 1, \ldots, r\}$, where each instance x_j has been assigned a class target from a small set of class targets $S = \{S_i | i = 1, \ldots, s\}$. Let $C_{\mathcal{G}} = \{C_i | i = 0, \ldots, n\}$ be the set of n level-0 classifiers C_i for a stacked generalizer $\mathcal{G}$. In this chapter, we utilize only two level-0 components ($n = 1$), one of which is the given base classifier C_0, which we assume has been fully trained. Denote by $C_i(x_j)$, for $x_j \in T$ and $C_i(x_j) \in S$, the class predicted by the classifier C_i when applied to x_j. The set of training instances T on which each C_i makes an error is denoted by

$$E(C_i, T) = \{x_j \in T | C_i \text{ } misclassifies \text{ } x_j\}.$$

The Coarse Reclassification algorithm is straightforward, as shown by the skeletal pseudocode given in Figure 4.3. As we discuss in Section 4.3, Coarse Reclassification is a member of a family of boosting algorithms that follow the more general pseudocode given in Figure 4.4

(page 141). Coarse Reclassification corresponds to $k = 0$ in that pseudocode. We describe the algorithm first here before showing how it fits into the more general framework in the next section.

In brief, the Coarse Reclassification uses prototype sampling to train a complementary component classifier on the training set. We show that this simple algorithm boosts or maintains nearest neighbor accuracy on a variety of data sets. The name "Coarse Reclassification" is derived from the complementary classifier's salient characteristics: it re-classifies the training data already classified by the base classifier, but does so by applying coarse hypotheses only.

To describe the algorithm, we examine in more detail the functions invoked in the pseudocode in Figure 4.3.

`sample-nearest-neighbor-classifiers`. To find a complementary nearest neighbor classifier C_1, we randomly sample m sets of s instances (with replacement) from the training set T, where s is the number of classes exposed in T. This algorithm was described in Section 3.4.1. One instance is drawn from each class. A nearest neighbor classifier is constructed using each sample of s instances as its prototype set. Each of these m classifiers is then used to classify the instances in T. The classifier that displays highest fitness function value, classification accuracy on T, is taken as the complementary classifier C_1. As a default, we take $m = 100$ samples. Since C_1 stores only as many prototypes as there are classes, for most data sets there are a very small number of decision regions in the hypotheses created by the sampled classifiers.

`train-stacked-generalizer`. A stacked generalizer $\mathcal{G}$ is trained on a level-1 training set that is created by applying the level-0 classifiers C_0 and C_1 to the (level-0) training set. For each original training instance $x \in T$ with class S_i, a level-1 training instance is created: $(C_0(x), C_1(x), S_i)$. The set of these three-tuples of class labels is the training set used to train the level-1 learning algorithm.

Experiment. To evaluate this algorithm, we start with a full nearest neighbor classifier whose accuracy we want to boost by incorporating it in a stacked generalizer. We also fix the level-1 learning algorithm as ID3, with the gain ratio feature selection metric. We took

Key:
C_0 : `level-0 base classifier`
C_1 : `level-0 complementary classifier`
L_1 : `level-1 classifier`
$\mathcal{G}$: `stacked generalizer`
T : `training set`
φ : `fitness function,` $\varphi(C_1, T)$ `is the accuracy of` C_1 `when applied to` T
p : `number of prototypes from each class`
m : `number of samples of prototype sets`
k : `cardinality of neighborhood (`k`-nearest neighbor)`

procedure `coarse-reclassification` $(C_0, T, L_1, \varphi, k, p, m)$
;;randomly sample prototypes (see Figure 3.3 for pseudocode)
$C_1 \leftarrow$ `sample-nearest-neighbor-classifiers` (T, φ, k, p, m)
$\mathcal{G} \leftarrow$ `train-stacked-generalizer` (T, L_1, C_0, C_1)

Figure 4.3: Pseudocode for the Coarse Reclassification algorithm.

$m = 100$ samples of prototypes, with one prototype per class, so $p = 1$. A 1-nearest neighbor algorithm was used, so that $k = 1$. The algorithm was run 50 times, consisting of five full 10-fold cross-validation runs. The average results are given in Table 4.3.

Analysis. Coarse Reclassification boosts nearest neighbor accuracy significantly at a significance level of $p \leq 0.05$ or better on four data sets (Breast Cancer Ljubljana, Diabetes, LED-7 Digit and Monks-2). On two of these data sets (Breast Cancer Ljubljana and Monks-2), the composite classifier is significantly more accurate than both the full nearest neighbor component classifier and the complementary component classifier. (We analyze the relative performance of the composite and complementary component classifiers for all the algorithms in Section 4.6.) The algorithm performed poorly on LED-24 Digit, Glass Recognition and Soybean data.

Coarse Reclassification demonstrates that a minimal classifier can be used to boost the accuracy of a full nearest neighbor classifier on a variety of these data sets. However, the poor performance of Coarse Reclassification on (at least) three data sets opens additional areas of investigation. For example, given that a full nearest neighbor classifier is a component in the

Table 4.3: Coarse Reclassification results. The second column gives the baseline nearest neighbor classifier accuracy. The third column gives the Coarse Reclassification accuracy. The fourth column gives the significance level at which the Coarse Reclassification algorithm accuracy is higher than the baseline nearest neighbor accuracy. The fifth column gives the average accuracy of the complementary component classifier added by the Coarse Reclassification algorithm. The sixth column gives the significance level at which the Coarse Reclassification accuracy is higher than that of the complementary component classifier.

Data	*NN*	*CR*	*Sig. NN*	*Cmpnt*	*Sig. Cpmnt*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	73.1 ± 6.9	0.020	71.4	0.002
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	97.4 ± 1.9	0.075	96.5	0.005
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	79.5 ± 7.1	0.147	79.9	0.848
Diabetes $_{760}$	68.7 ± 5.1	73.1 ± 5.1	0.019	72.9	0.254
Glass Recognition $_{210}$	73.3 ± 10.3	70.3 ± 9.9	0.992	53.3	0.000
Hepatitis $_{150}$	77.3 ± 11.4	80.2 ± 6.3	0.184	83.3	0.982
Iris Plants $_{150}$	91.3 ± 7.1	95.1 ± 3.8	0.178	95.8	0.938
LED-24 Digit $_{200}$	36.5 ± 9.4	27.2 ± 8.0	0.999	30.0	0.814
LED-7 Digit $_{500}$	71.2 ± 7.7	73.6 ± 7.8	0.008	73.3	0.381
Lymphography $_{140}$	80.7 ± 8.9	77.9 ± 10.7	0.860	74.0	0.062
Monks-2 $_{430}$	48.8 ± 4.7	67.5 ± 5.8	0.000	48.9	0.000
Promoter $_{100}$	79.0 ± 9.9	79.6 ± 7.0	0.367	65.0	0.000
Soybean $_{300}$	91.0 ± 3.9	87.1 ± 4.4	0.999	64.4	0.000

stacked generalizer, is there some way to ensure that the stacked generalizer is never less accurate than that classifier? For instance, if the composite classifier always output the prediction of the base classifier, performance no worse than the base classifier would be guaranteed. We investigate a more conservative version of Coarse Reclassification in Chapter 5, in the context of a more general discussion of class probability estimates.

4.3 Deliberate Misclassification

In this section we describe the Deliberate Misclassification algorithm for constructing a stacked generalizer to boost the accuracy of a given base classifier. This algorithm is designed to introduce diversity into the complementary component classifier by heuristic manipulation of the labels of the training instances.

An informal example may capture some intuition behind the algorithm. Suppose that you have just spotted your toddler about to put his finger into a live electric outlet. You want to train the child not to repeat this mistake. To ensure that the mistake is not made again, you go around the house pointing to electrical outlets, telling him “No, do not put your finger in here.” In addition to the live electric outlets in the house, you also warn him against some things that in fact would not harm him. An outlet that is not working is not a danger to him, but you tell him not to put his finger in the dead outlet. You point to the receptacle on an extension cord that is not plugged in and tell him “no”. In effect, you mis-label these examples in training the toddler. Mis-labeling examples that are close to an error can be a form of excessive training to assure that a particular error is not repeated.

The Deliberate Misclassification algorithm is heuristic and is based on such a simple intuition. If the goal is to create a complementary classifier that correctly classifies instances that the base classifier has misclassified, then the base classifier decision boundaries must be altered by the complementary classifier. This algorithm moves a decision boundary by changing the target classes of selected instances in a neighborhood of base classifier errors. This algorithm works to create diverse classifiers, which is an objective of composite classification even if the mis-classification costs are the same for all categories of errors.

From one standpoint, it is perverse to change class target labels. In this view, the underlying concept is defined extensionally through a presentation of the training instances. From another standpoint, it is not at all perverse, but an extension of common pre-processing methods. Data cleaning and data re-representation are preliminary phases of many learning algorithms. Outliers are identified and removed, features are selected, weighted, normalized, and transformed, and subsamples of the data are taken. From this standpoint, there is no *a priori* reason to believe that a particular presentation of a training set is the best one to learn an underlying concept or to extract regularities from the data. The instance class labels are an important part of that presentation, but one over which we also have control during the training phase. Instance re-labeling is, however, a new approach to creating diverse component classifiers that extends previous training set resampling techniques by manipulating class labels as well as drawing instances.

Our hypotheses are that (A) nearest neighbor accuracy can be boosted within a two-component boosting architecture created by Deliberate Misclassification, which trains a complementary classifier on a training set in which some of the original class labels have been changed, and (B) Deliberate Misclassification will outperform Coarse Reclassification. The null hypothesis corresponding to hypothesis B is that Deliberate Misclassification is not significantly more accurate than Coarse Reclassification.

The Deliberate Misclassification algorithm takes some time to describe, and so we summarize our results here first. As to hypothesis A, Deliberate Misclassification does succeed in boosting a nearest neighbor classifier, and in fact shows significant improvement over a nearest neighbor algorithm on one data set where Coarse Reclassification does not. It also shows significant improvement over a nearest neighbor classifier on more data sets (five) than Coarse Reclassification.

As to the hypothesis B, in Section 4.6, we show that on average across a number of ten-fold cross-validation runs, Deliberate Misclassification significantly outperforms Coarse Reclassification on only one data set, the Soybean data, and that is only at a low confidence level. On the Soybean data, however, in 45 pairwise comparisons of pairs of ten-fold cross validation runs of the two algorithms, the Deliberate Misclassification algorithm sometimes performs significantly more accurately ($p \leq 0.05$), and never significantly less accurately, than Coarse Reclassification. On this basis, Deliberate Misclassification is superior on this one data set. For all other data sets, though, from a statistical standpoint we accept the null hypothesis that Deliberate Misclassification is not more accurate than Coarse Reclassification. We discuss the reasons why Deliberate Misclassification does not significantly improve upon Coarse Reclassification in Section 4.6, where we compare the four boosting algorithms.

We now describe the Deliberate Misclassification algorithm. This algorithm retains the Coarse Reclassification characteristic that it constructs a small complementary classifier, but in addition it also changes the class targets of some instances near the errors committed by the base classifier. Pseudocode for the algorithm is given in Figure 4.4. The algorithm is outlined below, and has the same inputs and outputs as Coarse Reclassification. We then describe the main step, `modify-targets`, in more detail.

1. `modify-targets.` Modify the target classes of selected $x_i \in T$, resulting in a new training set, T'.

2. `sample-nearest-neighbor-classifiers.` Train a complementary classifier C_1 on the new training set, T', by sampling nearest neighbor classifiers.

3. `train-stacked-generalizer.` Train a stacked generalizer $\mathcal{G}$, consisting of a predetermined level-1 algorithm (ID3) and level-0 components C_0 and C_1, on the (original) training set T.

The pseudocode in Figure 4.4 is written to emphasize that Coarse Reclassification and Deliberate Misclassification are members of a family of algorithms. The family is indexed by the size (cardinality) of the neighborhood of each base classifier error in which instances are subject to relabeling. In the Coarse Reclassification algorithm, no labels are changed; in the Deliberate Misclassification algorithm implemented here, one neighbor of each training error may have its label changed.

We examine these procedures in more detail.

`modify-targets.` The instances whose class targets are to be changed are a subset of the instances that are near the errors made by the base classifier. These nearby instances are re-targeted with the correct class of the errors themselves. This sleight-of-hand creates a more class-homogeneous neighborhood around the errors. Since we apply local, instance-based classifiers that make predictions based on local class assignments, the intuition is that the base classifier errors are more likely to be correctly classified by the complementary classifier: now they live in a neighborhood of instances with the same class.

The `modify-targets` procedure in the Deliberate Misclassification algorithm has the following steps.

Key:
C_0 : level-0 base classifier
C_1 : level-0 complementary classifier
L_1 : level-1 classifier
$\mathcal{G}$: stacked generalizer
T : training set
T' : training set with (possibly) modified class labels
$T_E \subset T$: instances whose class labels will be changed
φ : fitness function, $\varphi(C_1, T')$ is the accuracy of C_1 when applied to T'
p : number of prototypes from each class
m : number of samples of prototype sets
k : cardinality of neighborhood (k-nearest neighbor)

procedure boost-by-prototype-sampling $(T, C_0, L_1, \varphi, k)$
$T' \leftarrow T$
$T' \leftarrow$ modify-targets (C_0, T, k)
;;randomly sample prototypes, use 1-nearest neighbor
$C_1 \leftarrow$ sample-nearest-neighbor-classifiers $(T', \varphi, 1, p, m)$
$\mathcal{G} \leftarrow$ train-stacked-generalizer (T, L_1, C_0, C_1)

procedure modify-targets (C_0, T, k)
$T_E \leftarrow \phi$
$E \leftarrow$ *create confusion matrix for* C_0 *on* T
;; see text for definition of maximally asymmetric
$E[A, B] \leftarrow$ *maximally asymmetric entry in* E
for each $a \in E[A, B]$
  do $U_a \leftarrow$ k-nearest-neighbors (a, T, k)
    for each $b \in U_a$
      do if $class(b) = B$
        then
          $class(b) \leftarrow A$
          $T_E \leftarrow T_E \cup \{b\}$
$T' \leftarrow T_E \cup (T - T_E)$

Coarse Reclassification:
boost-by-prototype-sampling $(T, C_0, L_1, \varphi, 0)$
Deliberate Misclassification:
boost-by-prototype-sampling $(T, C_0, L_1, \varphi, 1)$

Figure 4.4: Pseudocode for the boost-by-prototype-sampling family indexed by k, the cardinality of a neighborhood in which class labels may be changed.

1. *Create a confusion matrix for the base-classifier on the training set.* Compute the confusion matrix E made by the base classifier C_0 on a training set T. Ripley [1996] defines a confusion matrix to consist of entries e_{ij}, where

$$e_{ij} = pr(prediction\, j | class\ i)$$

the conditional probability that class j is predicted when the correct class is i. The confusion matrices we use have entries that are frequency counts, rather than probabilities. For each pair of classes i and j, e_{ij} records the number of entries in the training set that are actually of class i but are predicted to be of class j. The entries along the diagonal correspond to correctly predicted training instances.

   Our base classifier is assumed to be a nearest neighbor classifier that uses all training instances as prototypes. This nearest neighbor classifier trivially will classify the training set with 100% accuracy, since each instance is its own nearest neighbor. For this reason, leave-one-out cross-validation is used to compute the accuracy of the base nearest neighbor classifier.

2. *Find the maximally asymmetric boundary.* The Deliberate Misclassification algorithm modifies the class targets of a subset of the instances that are in a neighborhood (of cardinality k for a user-specified parameter k) of the errors made by the base classifier along the most *asymmetric* inter-class boundaries. For the boundary between classes A and B, denoted (A, B), the degree of asymmetry may be characterized by the difference between the number of errors made by the base classifier in one direction (*e.g.,* instances of class A misclassified as members of class B) and the number of errors in the opposite direction (*e.g.,* B's misclassified as A's).

   As we discuss in the next step, the reason that asymmetric boundaries are studied is to attempt to lower the risk that re-labeling will induce more errors than it corrects. When Deliberate Misclassification changes class labels to help ensure that instances of class A

are not mistakenly classified as class B, the risk is that it will cause instances of class B to be incorrectly classified as class A.

More formally, if E is the confusion matrix for the base classifier on the training set, a *maximally asymmetric* inter-class boundary $(X, Y)_{asymmetric}$ is given by:

$$(X, Y)_{asymmetric} = argmax_{(X,Y) \in S \times S}[E[X, Y] - E[Y, X]]$$

where S is the set of class labels. The algorithm also uses a constant default threshold $\delta > 0$, to require that difference $E[X, Y] - E[Y, X] \geq \delta$ to ensure that there actually is a positive imbalance of errors along a boundary. In the reported experiments, we use $\delta = 2$. In two-class problems, only one class boundary will be changed, since a positive asymmetry in errors will exist in only one direction for the two classes.

Example. Suppose there are three classes, A, B and C and the confusion matrix is:

		Predicted Class		
		A	B	C
	A	10	4	1
True Class	B	2	11	1
	C	2	0	12

where the rows specify the true class labels and the columns specify the class labels predicted by the base classifier. Thus four instances that were actually of class A were misclassified as class B. This entry, indexed by $[A, B]$, shows the largest difference with the entry that is its reflection across the main diagonal in the confusion matrix. See Table 4.4.

This boundary could be represented by the base classifier boundary in Figure 4.5, for example. In the figure, four instances from class A are mistakenly classified as B by the base classifier; two instances of class B are mistakenly classified as A.

3. *Modify targets of instances near the errors.* For each error identified in the previous step and a k fixed in advance, determine the k-nearest neighbors that have the class label *predicted*

Table 4.4: Differences between each confusion matrix and its opposite entry, *Opp.Entry*, the entry that is the reflection across the main diagonal.

Entry	*Opp.Entry*	*Entry Value*	*Opp.EntryValue*	*Diff.*
A,B	B,A	4	2	2
A,C	C,A	1	2	-1
B,A	A,B	2	4	-2
B,C	C,B	1	0	1
C,A	A,C	2	1	1
C,B	B,C	0	1	-1

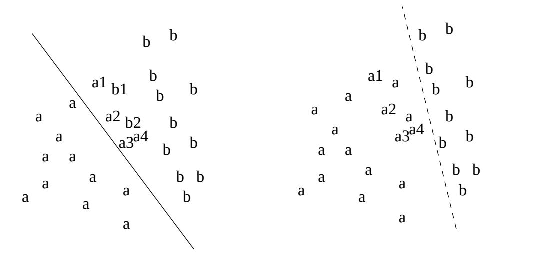

Figure 4.5: Shifting a decision boundary during Deliberate Misclassification. On the left is the boundary for the base classifier; on the right is the boundary for the complementary classifier. Instances below (to the left) of the boundary are predicted to be from class A. Instances above (to the right) of each decision boundary are classified B. The correct class of each instance is given by the label a for true class A and b for class B. The errors of the base classifier from class A are labeled $a_i, i = 1, \ldots, 4$. As a result, the instances labeled b_j, $j = 1, 2$ have their class targets changed from B to A in the training set for the complementary classifier.

by the base classifier. These neighbors are then re-assigned the true class of the instance misclassified by the base classifier.

Example. Suppose the most asymmetric class boundary is (A, B), that is, the greatest asymmetry of errors on the training set T arises by predicting B for instances whose true class is A. For each error of this type, compute its k nearest neighbors. (We used $k = 1$.) Each neighbor that is of class B is then re-labeled as class A (Figure 4.5).

Thus, the instances *near* near errors are re-targeted. The new class for these instances is the *actual class* of the error, the instance that was misclassified.

Informal Example. Since this step may be counter-intuitive, let us return for motivation to the informal example of the toddler and the electric socket. Suppose that the two classes are *dangerous* and *not dangerous* [1] . The live socket was *dangerous*. The child mistakenly classified the live socket as *not dangerous*. In order to train the child not to repeat that error, several examples *near* the error were re-labeled in this scenario. The true label for the dead-outlet example is *not dangerous*; it was re-labeled *dangerous*. The label for the extension cord that was not plugged into the wall was changed from *not dangerous* to *dangerous*. The new label for these two instances is the *actual class* of the example that was misclassified. The live outlet was misclassified, its actual class is *dangerous*, and so *dangerous* is the new label for these two instances.

This informal example also helps to show why we focus on asymmetric boundaries. Before we re-label examples, we would hope that the re-labeling will not cause more errors than it removes. Let us suppose that there are many more cases where the child wrongly predicts *not dangerous* than where the child wrongly predicts *dangerous*. In that case, we will not worry so much about changing some examples from *not dangerous* to *dangerous*. Even if a few more situations that are *not dangerous* are wrongly classified as a result of the re-labeling, we hope we will more than make up for that loss by correctly classifying a larger number of *dangerous* examples.

This informal example also points to another situation where instance re-labeling may be advisable: where the loss function for misclassifications is not uniform for examples of each class. Where the cost of making an error of one sort is much greater than the cost of making the opposite error, re-labeling might be applied fruitfully. In this example, the immediate cost of misclassifying an instance that is actually *dangerous* is much greater than misclassifying an instance that is *not dangerous*. Instances that are *not dangerous* could be re-labeled to reduce the number of high-cost misclassifications.

To summarize this re-labeling step, this sleight-of-hand creates a more class-homogeneous neighborhood around the errors. The intuition is that since we employ instance-based

[1] Readers familiar with tort law may be reminded of disputes about the meaning of "inherently dangerous" [Levi, 1949].

classifiers that make predictions based on local class assignments, the base classifier errors are more likely to be correctly classified by the complementary classifier: now they live in a neighborhood of instances with the same class.

The algorithm does not guarantee that only "helpful" instances will have their targets changed. However, by concentrating on asymmetric boundaries, the algorithm raises the possibility that a complementary classifier may correctly classify a larger number of instances at the expense of misclassifying a smaller number, resulting in a net gain in accuracy. Nevertheless, there is unhedged risk that changing class labels will introduce more errors than it removes. Even if this risk is realized, the Deliberate Misclassification algorithm may still be useful. The ultimate goal is not to create an accurate complementary classifier, but to create one that makes different errors and thereby contributes to a more accurate stacked generalizer.

4. *Add the re-labeled instances to the training set.* The union of (i) the training instances whose targets have not been modified and (ii) the training instances whose targets have been modified now constitutes a new training set T'. The training instances have the same attribute values as before; some of them simply have new targets.

The algorithm may also be extended so that the instances near more than one disputed boundary receive modified targets. For simplicity, we show the algorithm only for one boundary change, but changes along multiple boundaries is a conceptually simple extension. In this variation, the entries in the confusion matrix are sorted by degree of asymmetry and the boundaries reflected in the n most asymmetric entries are altered as by `modify-targets`.

Experiment. The Deliberate Misclassification algorithm was run with the following parameters: $m = 100$ samples, neighborhood size $k = 1$, $p = 1$ prototype per class, and two boundaries were changed. In other experiments, changing all the boundaries, rather than just the two most asymmetric, did not improve results significantly. As usual, C_0 is a full nearest neighbor classifier and L_1 is ID3. Ten-fold cross validation was performed. The entire 10-fold cross validation run was repeated nine times, resulting in 90 invocations of the algorithm. The algorithm was run a large number of times due to a large observed variance in accuracies in some cross-validation runs.

Analysis. Results are displayed in Table 4.5. Data headings are the same as those given for Coarse Reclassification in Table 4.3. We see $p \leq 0.05$ significant improvements on five data sets (Breast Cancer Ljubljana, Diabetes, Iris Plants, LED-7 Digit and Monks-2). In addition, the resulting stacked generalizer was significantly more accurate than the complementary component classifier on three of these five data sets. The Deliberate Misclassification algorithm also displays significantly higher accuracy than nearest neighbor on Iris Plants, whereas Coarse Reclassification does not.

Table 4.5: Deliberate Misclassification results.

Data	*NN*	*DM*	*Sig. NN*	*Cmpnt*	*Sig. Cmpnt*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	72.5 ± 6.9	0.046	69.4	0.016
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	97.3 ± 2.0	0.095	96.6	0.018
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	79.5 ± 4.4	0.085	79.0	0.174
Diabetes $_{760}$	68.7 ± 5.1	71.6 ± 4.4	0.040	69.8	0.009
Glass Recognition $_{210}$	73.3 ± 10.3	70.0 ± 10.0	0.969	54.1	0.000
Hepatitis $_{150}$	77.3 ± 11.4	79.5 ± 7.4	0.274	80.6	0.826
Iris Plants $_{150}$	91.3 ± 7.1	94.8 ± 3.3	0.018	95.4	0.820
LED-24 Digit $_{200}$	36.5 ± 9.4	27.7 ± 9.4	0.997	29.4	0.723
LED-7 Digit $_{500}$	71.2 ± 7.7	73.3 ± 7.3	0.015	72.1	0.150
Lymphography $_{140}$	80.7 ± 8.9	76.7 ± 11.7	0.922	72.6	0.040
Monks-2 $_{430}$	48.8 ± 4.7	65.8 ± 5.9	0.000	41.6	0.000
Promoter $_{100}$	79.0 ± 9.9	80.7 ± 7.1	0.218	63.8	0.000
Soybean $_{300}$	91.0 ± 3.9	87.9 ± 4.4	0.998	65.5	0.000

The number of instances changed per data set for these experiments is given in Table 4.6. For some data sets, a small number of instances were given new labels for training the complementary classifier. Since the number of instances changed depends partly on the number of errors committed by the base classifier, very few instances are changed for data sets that display high accuracy for the base nearest neighbor classifier (e.g., Breast Cancer Wisconsin, Iris Plants, Soybean).

Table 4.6: Average number of instances with changed targets and fraction of training set instance targets changed.

Data	*Changed*	*Fraction*
Breast Cancer Ljubljana $_{280}$	40.1	0.16
Breast Cancer Wisconsin $_{690}$	11.7	0.02
Cleveland Heart Disease $_{300}$	9.2	0.03
Diabetes $_{760}$	96.9	0.14
Glass Recognition $_{210}$	14.0	0.07
Hepatitis $_{150}$	5.0	0.04
Iris Plants $_{150}$	0.5	0.00
LED-24 Digit $_{200}$	8.4	0.04
LED-7 Digit $_{500}$	5.9	0.01
Lymphography $_{140}$	9.8	0.08
Monks-2 $_{430}$	50.8	0.13
Promoter $_{100}$	14.5	0.15
Soybean $_{300}$	11.3	0.04

The Deliberate Misclassification algorithm relies on a previously untried technique for boosting nearest neighbor accuracy. This implementation of Deliberate Misclassification showed significant improvement on five data sets in this collection. It remains for future research to determine the limits of this approach.

4.4 Composite Fitness

In the previous sections we considered two algorithms that try to maximize the accuracy of a complementary component classifier to select prototypes. Since *component* accuracy has been regarded by previous research as an important element of *composite* accuracy, maximizing component accuracy in the Coarse Reclassification algorithm was a reasonable approach, especially to establish baseline performance. The Deliberate Misclassification algorithm introduced a measure of diversity before trying to maximize component accuracy on a training set.

In the next two sections we abandon this indirect approach to maximizing composite accuracy and attempt instead to maximize the accuracy of the composite classifier directly. Although maximizing the accuracy of a classifier on the training set is a common technique, our survey in Chapter 2 revealed that component classifiers are frequently trained without determining their direct effect on composite accuracy. For example, in Schapire's boosting algorithm, the training regime for the components does not take account of the training accuracy of the composite classifier as a whole [Schapire, 1990]. The benefit of an approach that emphasizes the accuracy of the entire composite classifier is that we need not specify the characteristics of the complementary component that should be maximized. The qualities to be maximized may be unknown, and may vary from data set to data set in a manner that is not known, either.

In this section, we investigate a basic algorithm that appeals to composite accuracy, the Composite Fitness algorithm. The Composite Fitness algorithm is the same as for Coarse Reclassification with one substitution. The fitness function used to evaluate a component classifiers is the training accuracy of the entire stacked generalizer, rather than the accuracy of the component classifier. Pseudocode for the algorithm is given in Figure 4.6.

To make this substitution, it is necessary to train the composite classifier for each sampled nearest neighbor component classifier. This addition makes this algorithm more expensive than Coarse Reclassification, which does not require that the stacked generalizer be trained for each sampled set of prototypes. Once the composite classifier has been trained, both take the same time to apply, however.

Experiment. As usual, one prototype per class is sampled ($p = 1$), and we use ID3 as the level-1 learning algorithm (L_1). We take $m = 25$ prototype set samples, in part due to the expense of retraining the stacked generalizer for each sample and in part due to the characteristics of training composite accuracy that we discuss below. The algorithm was run 30 times, consisting of three complete ten-fold cross-validation runs. Average test accuracies are given in Table 4.7. Data headings are the same as those given for Coarse Reclassification in Table 4.3.

Analysis. The results of the Composite Fitness algorithm compared to the baseline nearest neighbor classifier are given in Table 4.7. We see significant improvement ($p \leq 0.05$) on four data sets (Breast Cancer Ljubljana, Cleveland Heart Disease, Iris Plants and Monks-2). The algorithm performs poorly on four data sets (Glass Recognition, LED-24 Digit, Lymphography

Key:

```
C_0 :  level-0 base classifier
C_j :  a coarse, complementary nearest neighbor classifier
L_1 :  level-1 classifier
G :  a stacked generalizer
T :  training set
S :  set of classes exposed in  T ,  S = {S_i | i = 1,...,s}
T_{S_i} :  training instances of class  S_i
P_j :  a set of prototypes
φ :  fitness function,  φ(G, T) , the accuracy of a stacked
  generalizer  G  when applied to  T
p :  number of prototypes from each class
m :  number of samples of prototype sets
k :  k-nearest neighbor

procedure composite-fitness (T, φ, k, p, m, L_1, C_0)
best-training-accuracy ← 0.0
best-stacked-generalizer ← nil
;; for each sample
loop for j from 1 to m
  loop for S_i in S
    ;; for each class, select prototypes
    append select-random-instances (p, T_{S_i})
    into P_j
  C_j ← create-nearest-neighbor-classifier (P_j, k)
  ;; create and train a stacked generalizer
  G ← train-stacked-generalizer (T, L_1, C_0, C_j)
  ;; the fitness function is training set accuracy of the stacked generalizer
  accuracy ← φ(G, T)
  when accuracy > best-training-accuracy
    best-training-accuracy ← accuracy
    best-stacked-generalizer ← G
finally return best-stacked-generalizer
```

Figure 4.6: Pseudocode for the Composite Fitness algorithm.

Table 4.7: Composite Fitness algorithm test accuracy.

Data	*NN*	*CF*	*Sig. NN*	*Cmpnt*	*Sig. Cmpnt*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	75.2 ± 7.5	0.002	65.6	0.006
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	97.2 ± 1.8	0.099	95.2	0.001
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	80.1 ± 4.6	0.042	80.7	0.974
Diabetes $_{760}$	68.7 ± 5.1	71.7 ± 4.6	0.058	72.5	0.943
Glass Recognition $_{210}$	73.3 ± 10.3	70.5 ± 9.9	0.939	40.0	0.000
Hepatitis $_{150}$	77.3 ± 11.4	78.7 ± 9.2	0.357	67.1	0.013
Iris Plants $_{150}$	91.3 ± 7.1	94.0 ± 5.8	0.009	91.3	0.090
LED-24 Digit $_{200}$	36.5 ± 9.4	26.8 ± 6.7	0.998	27.3	0.565
LED-7 Digit $_{500}$	71.2 ± 7.7	72.7 ± 7.6	0.115	49.5	0.000
Lymphography $_{140}$	80.7 ± 8.9	76.4 ± 10.6	0.973	58.6	0.004
Monks-2 $_{430}$	48.8 ± 4.7	64.1 ± 7.3	0.000	47.4	0.001
Promoter $_{100}$	79.0 ± 9.9	80.3 ± 6.7	0.321	65.7	0.000
Soybean $_{300}$	91.0 ± 3.9	86.7 ± 4.9	0.999	58.7	0.000

and Soybean). The algorithm increases or maintains base-level accuracy on nine of the thirteen data sets.

On three data sets (Cleveland Heart Disease, Diabetes and LED-24 Digit), the composite classifiers created by Composite Fitness were less accurate than the respective component classifiers. On all the other data sets, the composite classifiers were significantly more accurate than the complementary component classifier they incorporate.

Given the arguments in favor of using the training accuracy of the composite classifier as a fitness function, we try to explain why training accuracy may not have been a good indicator of composite accuracy on the four data sets where the results were poor.

We suggest two reasons for these failures. One suggestion is that there is little correlation between the training and test accuracy of the stacked generalizers. In this situation, accuracy on the training set does not provide an adequate indication of a stacked generalizer's accuracy when applied to unseen test cases. A second suggestion is that maximizing training accuracy sometimes permits small differences in training accuracy to influence the search for prototypes. These small differences in the fitness function value may be inconsequential or even random. In turn, there may be little variation in training accuracy because, on some data sets, the base classifier's accuracy can damp the accuracy of the entire composite classifier. Where the complementary classifier displays lower accuracy than the base nearest neighbor classifier, the prediction of the base classifier tends to dominate composite predictions and helps to stabilize and maintain the accuracy of the entire stacked generalizer.

To test these explanations, we performed an experiment in which 100 stacked generalizers were built using a full nearest neighbor classifier and a minimal nearest neighbor classifier as components. The prototypes in the minimal classifier were sampled at random, according to our usual sampling plan. We then compared the training and test accuracy on each data set. Composite Fitness performed poorly on the Soybean data and Figure 4.7 shows the relationship between composite *training* and composite *test* accuracy on this data. Both effects we suggested can be seen. (1) There is little correlation between the training and test accuracy of the stacked generalizers. Also, (2) the training accuracy falls within a small range, smaller than test accuracy.

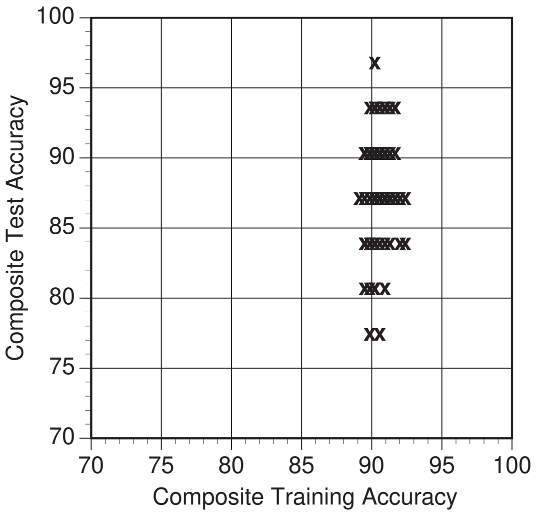

Figure 4.7: Relationship between the training and testing accuracy of random stacked generalizers on the Soybean data.

Table 4.8 repeats the first four columns from Table 4.7 and also gives the statistical correlation between the training and test accuracy of the stacked generalizers just described. The table shows that where the Composite Fitness algorithm performs poorly (Glass Recognition, LED-24 Digit, Lymphography and Soybean), the correlation between composite training and test accuracy is low or negative, as predicted. On the other hand, there are two data sets where the algorithm performs well (Cleveland Heart Disease and Iris Plants), where the train-test correlation is low or even negative! High train-test correlation is apparently not a necessary condition for success with this algorithm.

Table 4.8: Composite Fitness algorithm test accuracy.

Data	*NN*	*CF*	*Sig.-NN*	*Correl.*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	75.2 ± 7.5	0.002	0.77
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	97.2 ± 1.8	0.099	0.70
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	80.1 ± 4.6	0.042	0.07
Diabetes $_{760}$	68.7 ± 5.1	71.7 ± 4.6	0.058	0.75
Glass Recognition $_{210}$	73.3 ± 10.3	70.5 ± 9.9	0.939	0.12
Hepatitis $_{150}$	77.3 ± 11.4	78.7 ± 9.2	0.357	0.42
Iris Plants $_{150}$	91.3 ± 7.1	94.0 ± 5.8	0.009	-0.20
LED-24 Digit $_{200}$	36.5 ± 9.4	26.8 ± 6.7	0.998	-0.06
LED-7 Digit $_{500}$	71.2 ± 7.7	72.7 ± 7.6	0.115	0.15
Lymphography $_{140}$	80.7 ± 8.9	76.4 ± 10.6	0.973	-0.25
Monks-2 $_{430}$	48.8 ± 4.7	64.1 ± 7.3	0.000	0.51
Promoter $_{100}$	79.0 ± 9.9	80.3 ± 6.7	0.321	0.22
Soybean $_{300}$	91.0 ± 3.9	86.7 ± 4.9	0.999	0.03

The accuracies achieved by the Composite Fitness algorithm on the Breast Cancer Ljubljana, Cleveland Heart Disease and Glass Recognition data are the highest achieved by any of the four boosting algorithms. All four algorithms failed to boost accuracy on the Glass Recognition data, however.

We now turn to the last algorithm in our study. The Composite Fitness–Feature Selection algorithm tries to maximize the accuracy of the entire stacked generalizer on the training set, but searches for a complementary nearest neighbor classifier that considers only a single feature. The objective of the Composite Fitness–Feature Selection algorithm is to increase the diversity of the component classifiers over that attained in the Composite Fitness algorithm.

4.5 Composite Fitness–Feature Selection

We show that a variation of the Composite Fitness algorithm achieves results comparable to the other boosting algorithms, but does so by creating a complementary component that is never more accurate than the base nearest neighbor classifier. The complementary component is typically very inaccurate.

The Composite Fitness–Feature Selection algorithm follows the same procedure as the Composite Fitness algorithm. It samples nearest neighbor component classifiers by sampling prototypes and tries to maximize composite accuracy. The difference is that Composite Fitness–Feature Selection samples component nearest neighbor classifiers that use *only one* feature in their distance metric.

Procedurally, there are two loops in the algorithm. In the outer loop, a prototype set is sampled, as in the other algorithms. In an inner loop, for each sampled prototype set, an exhaustive search is done for the "best" feature. Here, the "best" feature is the one that, when used in the distance metric for the complementary component classifier, maximizes the training accuracy of the entire composite classifier. The exhaustive search is the first step of a Forward Sequential Selection algorithm, a classical statistical algorithm for feature selection [Krzanowski, 1988]. So, for each sampled prototype set, for each feature, a stacked generalizer is created, trained and applied to the training set. Pseudocode for the algorithm is given in Figure 4.8.

In Figure 4.8, `create-k-nn-classifier-for-features` creates a nearest neighbor classifier that applies a modified distance metric d_f, which defines the distance between two instances as the distance between two instances on a single feature. This modified metric is defined as follows. The metric is just the restriction of our usual distance metric to a single feature.

Let $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ be instances, and let $f = f_j$ be the jth feature. If f is a numeric feature then the distance between x and y, $d(x, y)$ is

$$d(x, y) = d_f(x, y) = |x_j - y_j|.$$

If f is a symbolic feature, where d_f is an exact-match metric for symbolic features:

$$d(x, y) = d_f(x_j, y_j) = \begin{cases} 0 & \text{if } x_j = y_j \\ 1 & \text{otherwise} \end{cases}$$

This is of course a very crude distance metric, especially where f is a symbolic feature. Our goal here is not to maximize accuracy, but to try to increase diversity.

Key:

```
C_0 :  level-0 base classifier
C_j :  a coarse, complementary nearest neighbor classifier
L_1 :  level-1 classifier
G :  a stacked generalizer
T :  training set
F :  set of features for instances in T, F = {f_l | l = 1,...,q}
S :  set of classes exposed in T, S = {S_i | i = 1,...,s}
T_{S_i} :  training instances of class S_i
P_j :  a set of prototypes
φ :  fitness function, φ(G, T), the accuracy of a stacked
  generalizer G when applied to T
p :  number of prototypes from each class
m :  number of samples of prototype sets
k :  k-nearest neighbor
```

```
procedure composite-fitness-feature-selection (T, φ, k, p, m, L_1, C_0)
best-training-accuracy ← 0.0
best-stacked-generalizer ← nil
;; for each sample
loop for j from 1 to m
  loop for S_i in S
    ;; for each class, select prototypes
    append select-random-instances (p, T_{S_i})
    into P_j
  loop for f_l in F
    C_j ← create-k-nn-classifier-with-features (P_j, k, {f_l})
    ;; create and train a stacked generalizer
    G ← train-stacked-generalizer (T, L_1, C_0, C_j)
    ;; the fitness function is training set accuracy of the stacked generalizer
    accuracy ← φ(G, T)
    when accuracy > best-training-accuracy
      best-training-accuracy ← accuracy
      best-stacked-generalizer ← G
finally return best-stacked-generalizer
```

Figure 4.8: Pseudocode for the Composite Fitness–Feature Selection algorithm.

Experiment. Preliminary experiments suggested that the features had a greater impact on stacked generalizer accuracy than the prototypes selected. In this experiment we took only five samples of prototypes, one from each class, and for each prototype set, an exhaustive search was done of the individual features. Since the median number of features is nine, the median number of classifiers considered for each run of the algorithm is 45; the minimum was 20 and the maximum was 290. The algorithm was run 30 times, consisting of three full ten-fold cross validation runs. Our results are averages over the three ten-fold runs. Table 4.9 gives the results of the Composite Fitness–Feature Selection algorithm against the nearest neighbor baseline. Data headings are the same as those given for Coarse Reclassification in Table 4.3.

Table 4.9: Composite Fitness–Feature Selection results.

Data	*NN*	*CF-FS*	*Sig. NN*	*Cmpnt*	*Sig. Cmpnt*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	74.6 ± 8.8	0.005	49.6	0.010
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	96.7 ± 1.7	0.217	87.4	0.000
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	77.7 ± 4.5	0.339	70.7	0.124
Diabetes $_{760}$	68.7 ± 5.1	73.7 ± 5.5	0.015	63.6	0.009
Glass Recognition $_{210}$	73.3 ± 10.3	69.0 ± 7.5	0.898	21.4	0.000
Hepatitis $_{150}$	77.3 ± 11.4	82.7 ± 10.5	0.076	26.7	0.000
Iris Plants $_{150}$	91.3 ± 7.1	93.3 ± 5.4	0.197	91.3	0.328
LED-24 Digit $_{200}$	36.5 ± 9.4	36.7 ± 10.0	0.474	12.2	0.000
LED-7 Digit $_{500}$	71.2 ± 7.7	72.6 ± 8.1	0.055	14.4	0.000
Lymphography $_{140}$	80.7 ± 8.9	76.7 ± 7.7	0.921	16.9	0.000
Monks-2 $_{430}$	48.8 ± 4.7	61.9 ± 6.2	0.000	43.7	0.004
Promoter $_{100}$	79.0 ± 9.9	82.5 ± 7.9	0.105	65.0	0.001
Soybean $_{300}$	91.0 ± 3.9	87.7 ± 5.0	0.998	5.0	0.000

Analysis. Table 4.9 shows that the Composite Fitness–Feature Selection algorithm is significantly more accurate than a full nearest neighbor classifier on three data sets at the $p \leq 0.05$ level (Breast Cancer Ljubljana, Diabetes and Monks-2). Composite Fitness–Feature Selection is the only one of the four algorithms that even maintains the accuracy of the base classifier on the LED-24 Digit data.

For every data set except Iris Plants and Cleveland Heart Disease, the composite classifier is significantly more accurate than the complementary component classifier. For example, on the Breast Cancer Ljubljana data set, on average a classifier with 49.6% accuracy was used to boost the accuracy of a nearest neighbor classifier from 67.5% to 74.6%. This algorithm demonstrates that highly inaccurate classifiers can be used to boost nearest neighbor accuracy on a variety of data sets.

4.6 Comparison of the Four Algorithms

We have proposed and evaluated four algorithms for boosting a nearest neighbor classifier. In the previous four sections, we compared the accuracy of the stacked generalizer created by each algorithm to a base nearest neighbor classifier. In this section, we compare the boosting algorithms to each other.

Our objectives are to answer the questions, *How well do these algorithms boost, when compared to each other?* and *How do these algorithms perform boosting?*. In Section 4.6.1, we compare how well they work in terms of composite accuracy. Our hypothesis is that these four algorithms create stacked generalizers that yield comparable accuracy. In Section 4.6.2, we compare how they work. Our hypothesis is that, even if the four algorithms may achieve similar accuracies, the algorithms are different in the *manner* they accomplish boosting. In particular, they create component classifiers that exhibit different degrees of accuracy and diversity.

4.6.1 Composite Accuracy

Based on our experiments with the four algorithms, it appears that the four boosting algorithms yield quite similar composite accuracy for many data sets. Table 4.10 summarizes these results.

The accuracies obtained by these four algorithms are often quite close. More than one algorithm showed a significant improvement over nearest neighbor on five data sets: Breast Cancer Ljubljana, Diabetes, Iris Plants, LED-7 Digit and Monks-2. All of these data sets are have two or three classes. The accuracy of these algorithms was higher than nearest neighbor on all the two-class problems (Breast Cancer Ljubljana, Breast Cancer Wisconsin, Cleveland Heart Disease, Hepatitis, Monks-2 and Promoter). In general, based on this collection of data sets, these boosting algorithms can be recommended for application to two-class problems.

Table 4.10: Summary of the accuracy obtained by the four boosting algorithms. Significant improvements in accuracy over a nearest neighbor classifier at the $p \leq 0.05$-level (or better) are flagged with an asterisk.

Data	*NN*	*CR*	*DM*	*CF*	*CF-FS*
Breast Cancer Ljubljana $_{280}$	67.5	73.1*	72.5*	75.2*	74.6*
Breast Cancer Wisconsin $_{690}$	96.2	97.4	97.3	97.2	96.7
Cleveland Heart Disease $_{300}$	77.0	79.5	79.5	80.1*	77.7
Diabetes $_{760}$	68.7	73.1*	71.6*	71.7	73.7*
Glass Recognition $_{210}$	73.3	70.3	70.0	70.5	69.0
Hepatitis $_{150}$	77.3	80.2	79.5	78.7	82.7
Iris Plants $_{150}$	91.3	95.1	94.8*	94.0*	93.3
LED-24 Digit $_{200}$	36.5	27.2	27.7	26.8	36.7
LED-7 Digit $_{500}$	71.2	73.6*	73.3*	72.7	72.6
Lymphography $_{140}$	80.7	77.9	76.7	76.4	76.7
Monks-2 $_{430}$	48.8	67.5*	65.8*	64.1*	61.9*
Promoter $_{100}$	79.0	79.6	80.7	80.3	82.5
Soybean $_{300}$	91.0	87.1	87.9	86.7	87.7

The algorithms show the poorest performance on the same four data sets: Glass Recognition, LED-24 Digit, Lymphography and Soybean. All of these problematic data sets have more than three classes: Glass Recognition (6 classes), LED-24 Digit (10), Lymphography (4) and Soybean (19). Individually, some of these data sets also have characteristics that may make them resistant to the four boosting algorithms. LED-24 Digit has 17 irrelevant features, and irrelevant features often pose difficulties for nearest neighbor classifiers that do not weight features. LED-24 Digit also has noise in the attribute values. As we discuss below, the Soybean data set has 712 missing symbolic feature values. Our algorithm for supplying missing symbolic values is simply to impute the most frequently occurring value for the feature across the training set, which is a naive technique. In his extensive comparison of data sets, Zheng [1993] characterizes the *predictive accuracy* of Lymphography as *Low*, indicating that the domain is difficult. Zheng defines the default *predictive accuracy* as the accuracy achieved by the baseline algorithm C4.5, which was 78.4% on the Lymphography data. The predictive accuracy of Glass Recognition (68.1%) and LED-24 Digit (64.0%) were also *Low*, according to this characterization.

In Section 4.8, we show that these four data sets have the smallest numbers of instances per class of the 13 data sets used. In that section, we further analyze the failures and successes of these algorithms, after we gather more data on their performance and the performance of their components in Section 4.6.2. However, based on this set of experiments, these algorithms cannot be recommended for multi-class problems, in general. Performance on the 10-class LED-7 Digit data was quite good, on the other hand.

Next, we treat the Coarse Reclassification algorithm as providing baseline performance and then compare the Deliberate Misclassification, Composite Fitness and Composite Fitness–Feature Selection algorithms to that baseline. The Coarse Reclassification algorithm is a convenient baseline since it simply tries to maximize component accuracy, which our literature survey revealed as probably the most important influence on composite accuracy.

Comparison to Deliberate Misclassification. Since the Deliberate Misclassification algorithm is identical to the Coarse Reclassification algorithm except for the modification of class labels, additional analysis is warranted to determine if this modification effects any change in composite accuracy.

Since our main objective in this chapter is to boost a base nearest neighbor classifier, this comparison is of secondary importance. On the primary boosting task, we showed in Section 4.3 that Deliberate Misclassification gives significantly higher accuracy than nearest neighbor on one data set (Iris Plants) on which Coarse Reclassification does not.

A second way to compare the two algorithms is by a head-to-head comparison of average accuracies computed on the basis of multiple cross-validation runs. Five ten-fold cross validation runs of Coarse Reclassification and nine ten-fold cross validation runs of Deliberate Misclassification were performed. Thus the Coarse Reclassification algorithm was run 50 times in total; the Deliberate Misclassification algorithm, 90 times. Deliberate Misclassification was run more times because a large variance in composite accuracy was observed for some data sets. Table 4.11 compares the composite accuracies obtained by these two algorithms.

There is little difference between the average accuracies of the two algorithms. There is evidence that Deliberate Misclassification is more accurate on Soybean data. With 13

Table 4.11: Coarse Reclassification accuracy compared to Deliberate Misclassification accuracy. The first column gives the data set. The second column gives the mean accuracy of Coarse Reclassification. The third column gives the mean accuracy of the Deliberate Misclassification algorithm. The fourth column gives the significance that Deliberate Misclassification has higher average accuracy than Coarse Reclassification.

Data	*CR*	*DM*	*Sig.*
Breast Cancer Ljubljana $_{280}$	73.1 ± 6.9	72.5 ± 6.9	0.755
Breast Cancer Wisconsin $_{690}$	97.4 ± 1.9	97.3 ± 2.0	0.647
Cleveland Heart Disease $_{300}$	79.5 ± 7.1	79.5 ± 4.4	0.516
Diabetes $_{760}$	73.1 ± 5.1	71.6 ± 4.4	0.932
Glass Recognition $_{210}$	70.3 ± 9.9	70.0 ± 10.0	0.624
Hepatitis $_{150}$	80.2 ± 6.3	79.5 ± 7.4	0.737
Iris Plants $_{150}$	95.1 ± 3.8	94.8 ± 3.3	0.638
LED-24 Digit $_{200}$	27.2 ± 8.1	27.7 ± 9.4	0.408
LED-7 Digit $_{500}$	73.6 ± 7.8	73.3 ± 7.3	0.697
Lymphography $_{140}$	77.9 ± 10.7	76.7 ± 11.7	0.768
Monks-2 $_{430}$	67.5 ± 5.8	65.8 ± 5.9	0.887
Promoter $_{100}$	79.6 ± 7.0	80.7 ± 7.1	0.240
Soybean $_{300}$	87.1 ± 4.4	87.9 ± 4.4	0.090

data sets, one would expect a significant improvement at the $p \leq 0.10$ level on at least one data set, due to purely random effects. The difference in accuracy is also extremely small. To investigate this comparison further, we take the 45 pairs of ten-fold cross validation runs and on each pair, compute the statistical significance that Deliberate Misclassification outperforms Coarse Reclassification. On Soybean data, the Deliberate Misclassification algorithm outperforms Coarse Reclassification significantly ($p \leq 0.05$) in almost 25% of the pairwise comparisons (10 of 45). Further, Coarse Reclassification never significantly outperforms Deliberate Misclassification on that data set, so the Deliberate Misclassification algorithm is evidently superior. On Soybean data, the added expense of changing class labels is apparently justified.

We have been unable to isolate the characteristics of the Soybean data that account for the slightly superior performance of Deliberate Misclassification. However, the Soybean data set does possess at least three characteristics that distinguish it from the other data sets. Of the 13 data sets we consider, it has the largest percentage of missing feature values (6.6%), the greatest number of classes (19) and the smallest average number of training instances per class (14.2). Thus, the Soybean data set is extreme along a number of dimensions.

Further, in a documentary file distributed with the U.C.I. Soybean data (named "soybean-explanation") [Murphy and Aha, 1994], a discussion between persons familiar with soybean diseases reveals some of the features may have incorrect values in the data file. These incorrect features in turn are said to affect some of the class labels. Although this evidence is anecdotal, if Deliberate Misclassification implicitly corrected some of the class labels on Soybean data, higher component and composite accuracy would result, as we observed.

Comparison to Composite Fitness and Composite Fitness–Feature Selection. We now compare the Coarse Reclassification algorithm to the Composite Fitness and Composite Fitness–Feature Selection algorithms. Table 4.12 shows that Breast Cancer Ljubljana is the only data set where Composite Fitness is significantly better than Coarse Reclassification. Composite Fitness–Feature Selection is significantly better than Coarse Reclassification on LED-24 Digit and Promoter data. Composite Fitness–Feature Selection is the only boosting algorithm of the four that even matched nearest neighbor accuracy on the LED-24 Digit data. The Composite Fitness and Composite Fitness–Feature Selection algorithms showed lower accuracy than Coarse Reclassification on Monks-2 data, though.

This analysis demonstrates that the accuracies of the stacked generalizers created by Deliberate Misclassification, Composite Fitness and Composite Fitness–Feature Selection are comparable to those created by Coarse Reclassification, with some exceptions. Nonetheless, the exceptions demonstrate that the algorithms do perform somewhat differently on the primary task of boosting a nearest neighbor classifier. Of course, these exceptions also may be important for classifier systems in fielded applications.

In the next section, we examine the characteristics of the complementary component classifiers created by each of the four algorithms to determine whether the *manner* in which they boost accuracy — or fail to boost accuracy — is nevertheless different.

Table 4.12: Composite Fitness and Composite Fitness–Feature Selection compared to Coarse Reclassification. The fourth column contains the significance level at which the Composite Fitness algorithm is higher than Coarse Reclassification. The sixth column contains the comparable value for the Composite Fitness–Feature Selection algorithm. Significant differences are marked with an asterisk.

Data	*CR*	*CF*	*Sig. CF*	*CF-FS*	*Sig. CF-FS*
Breast Cancer Ljubljana $_{280}$	73.1 ± 6.9	75.2 ± 7.5	0.028*	74.6 ± 8.8	0.157
Breast Cancer Wisconsin $_{690}$	97.4 ± 1.9	97.2 ± 1.8	0.675	96.7 ± 1.7	0.915
Cleveland Heart Disease $_{300}$	79.5 ± 7.1	80.1 ± 4.6	0.312	77.7 ± 4.5	0.811
Diabetes $_{760}$	73.1 ± 5.1	71.7 ± 4.6	0.912	73.7 ± 5.5	0.332
Glass Recognition $_{210}$	70.3 ± 9.9	70.5 ± 9.9	0.443	69.0 ± 7.5	0.657
Hepatitis $_{150}$	80.2 ± 6.3	78.7 ± 9.2	0.835	82.7 ± 10.5	0.167
Iris Plants $_{150}$	95.1 ± 3.8	94.0 ± 5.8	0.804	93.3 ± 5.4	0.884
LED-24 Digit $_{200}$	27.2 ± 8.1	26.8 ± 6.7	0.588	36.7 ± 10.0	0.002*
LED-7 Digit $_{500}$	73.6 ± 7.8	72.7 ± 7.6	0.919	72.6 ± 8.1	0.933
Lymphography $_{140}$	77.9 ± 10.7	76.4 ± 10.6	0.855	76.7 ± 7.7	0.684
Monks-2 $_{430}$	67.5 ± 5.8	64.1 ± 7.3	0.989	61.9 ± 6.2	0.992
Promoter $_{100}$	79.6 ± 7.0	80.3 ± 6.7	0.356	82.5 ± 7.9	0.038*
Soybean $_{300}$	87.1 ± 4.4	86.7 ± 4.9	0.670	87.7 ± 5.0	0.235

4.6.2 Component Diversity and Accuracy

In this section, we examine the general hypothesis that the manner is different in which the four algorithms boost accuracy. Specifically, we show that the component classifiers created by the four algorithms often reflect different combinations of accuracy and diversity, even where they yield nearly equal composite accuracy.

This conclusion is demonstrated in three parts of this subsection. For each of the boosting algorithms, we analyze (1) component diversity, (2) component accuracy and (3) the combination of diversity and accuracy. We appeal to the components created by Coarse Reclassification as a baseline, since that algorithm made no attempt to maximize component diversity, and did attempt to maximize component accuracy.

Diversity. An examination of component diversity in isolation provides the first piece of evidence that the manner is different in which the algorithms are working. Table 4.13 records the diversity on the test set of the component classifier created by each algorithm with respect to the base nearest neighbor classifier. Diversity is defined as the fraction of the test set for which the two component classifiers make different classifications but where one of them gives the correct classification. The values shown are the averages over all runs of the algorithms, except for the Composite Fitness–Feature Selection data, which are averages over one 10-fold cross-validation run.

Table 4.13: Component diversity on the test set.

Data	*CR*	*DM*	*CF*	*CF-FS*
Breast Cancer Ljubljana $_{280}$	0.25	0.31	0.36	0.37
Breast Cancer Wisconsin $_{690}$	0.03	0.03	0.04	0.21
Cleveland Heart Disease $_{300}$	0.19	0.18	0.18	0.56
Diabetes $_{760}$	0.27	0.31	0.28	0.36
Glass Recognition $_{210}$	0.38	0.39	0.47	0.56
Hepatitis $_{150}$	0.17	0.17	0.32	0.49
Iris Plants $_{150}$	0.06	0.06	0.09	0.08
LED-24 Digit $_{200}$	0.38	0.39	0.34	0.39
LED-7 Digit $_{500}$	0.12	0.14	0.37	0.63
Lymphography $_{140}$	0.29	0.29	0.42	0.70
Monks-2 $_{430}$	0.46	0.53	0.60	0.72
Promoter $_{100}$	0.34	0.34	0.41	0.37
Soybean $_{300}$	0.30	0.31	0.38	0.88

An initial observation is that for 8 of the 13 data sets, there is a non-decreasing sequence of diversity of the four algorithms in this table. Although the algorithms were not designed to do so, the four algorithms create increasingly more diverse component classifiers, in general. Since Coarse Reclassification emphasized component accuracy only, and did not take diversity into consideration, the components created by that algorithm were the least diverse of the four boosting algorithms. Briefly, we compare the other three algorithms to Coarse Reclassification on the basis of their component diversity, and make the following observations.

Deliberate Misclassification. The differences in diversity between Coarse Reclassification and Deliberate Misclassification are small, in general. These small differences in diversity were observed despite Deliberate Misclassification's relabeling of class targets. There are a number of possible explanations. For example, the model class may have been too coarse to be affected by the changes in the class labels, the wrong labels were changed, or insufficiently many labels were changed. An additional explanation goes to the measure of diversity itself. Two composite classifiers with the same levels of component diversity can have different test-set performance, nevertheless.

Composite Fitness. Composite Fitness yielded more diverse components than Coarse Reclassification on 11 of 13 data sets. Recall that the Coarse Reclassification algorithm attempted to find the most accurate complementary component, and Composite Fitness allowed the balance between accuracy and diversity be determined implicitly, reflected in composite training accuracy. On all but two data sets, the balance between the two factors was resolved by Composite Fitness in favor of components that showed more diversity (and lower accuracy, as we shall see) than those created by Coarse Reclassification.

Composite Fitness–Feature Selection. In general, the most diverse classifiers were created by the Composite Fitness–Feature Selection algorithm. This algorithm searched for complementary component classifiers in a model class that applied only one feature. This model class was intentionally selected to increase diversity, and increased diversity did result, in fact.

In summary, although Coarse Reclassification and Deliberate Misclassification displayed similar diversity, an inspection of the component diversity shows that three of the algorithms are boosting in different ways.

Accuracy. Next, we inspect component accuracy in order to determine the extent to which the four boosting algorithms are working differently with respect to component accuracy. We do this in two ways. First, we comment briefly on the component accuracies, as we did for diversity. Second, where a composite classifier boosted successfully, we compare the accuracy of that composite classifier with the accuracy of the complementary component classifier.

First, for most data sets, Table 4.14 shows a small difference between the accuracies of the components created by the Coarse Reclassification and Deliberate Misclassification algorithms. The values shown are the averages over all runs of the algorithms, except for the Composite Fitness–Feature Selection data, which are averages over one 10-fold cross-validation run. In general, the components created by Composite Fitness were less accurate than those added by Coarse Reclassification. Except for the Promoter and Iris Plants data sets, the components created by Composite Fitness–Feature Selection were substantially less accurate than those created by Coarse Reclassification and the other algorithms.

Table 4.14: Nearest neighbor and component test accuracy. Nearest neighbor accuracy is here for comparison.

Data	*NN*	*CR*	*DM*	*CF*	*CF-FS*
Breast Cancer Ljubljana $_{280}$	67.5	71.4	69.4	65.6	61.0
Breast Cancer Wisconsin $_{690}$	96.2	96.5	96.6	95.2	78.3
Cleveland Heart Disease $_{300}$	77.0	79.9	79.0	80.7	44.7
Diabetes $_{760}$	68.7	72.9	69.8	72.5	65.8
Glass Recognition $_{210}$	73.3	53.3	54.1	40.0	29.5
Hepatitis $_{150}$	77.3	83.3	80.6	67.1	49.3
Iris Plants $_{150}$	91.3	95.8	95.4	91.3	94.0
LED-24 Digit $_{200}$	36.5	30.0	29.4	27.3	12.0
LED-7 Digit $_{500}$	71.2	73.3	72.1	49.5	18.0
Lymphography $_{140}$	80.7	74.0	72.6	58.6	21.4
Monks-2 $_{430}$	48.8	48.9	41.6	47.4	44.7
Promoter $_{100}$	79.0	65.0	63.8	65.7	66.0
Soybean $_{300}$	91.0	64.4	65.5	58.7	5.3

Second, in order to show unalloyed improvement through classifier combination, ideally it would be shown that a composite classifier is significantly more accurate than all of its components. (On the other hand, in boosting a base classifier, an algorithm would be successful if it created a component classifier with accuracy statistically indistinguishable from the base in order to yield a significantly more accurate composite classifier.)

To track this ideal measure of success, as each algorithm was evaluated in the previous sections, we showed the average accuracy of the complementary component and the significance of the improvement in accuracy of the composite classifier over that component (Tables 4.3, 4.5, 4.7 and 4.9). These results are summarized in Table 4.15, which gives the number of times that each boosting algorithm shown significant improvement over both the base nearest neighbor classifier and the complementary component classifier.

Table 4.15 shows that the Composite Fitness–Feature Selection algorithm creates a composite that is significantly more accurate than the complementary component for every data

Table 4.15: Summary of stacked generalizer accuracy by four boosting algorithms. *SG>NN* denotes the number of data sets out of 13 on which the stacked generalizer created by a boosting algorithm was significantly more accurate than the base nearest neighbor classifier. *SG>Component* denotes the number of the data sets listed in the previous column on which the stacked generalizer also displayed accuracy higher than the complementary component classifier created by a boosting algorithm.

Algorithm	*SG>NN*	*SG>Component*
Coarse Reclassification	4	2
Deliberate Misclassification	5	3
Composite Fitness	4	2
Composite Fitness–Feature Selection	3	3

set in which it boosted accuracy successfully. The other three algorithms create composite classifiers that are more accurate than the component 50% or 60% of the time.

In summary, several differences among the four algorithms with respect to the accuracy of the components may be seen. Composite Fitness and Composite Fitness–Feature Selection frequently create less accurate components than Coarse Reclassification. In our experiments, Composite Fitness–Feature Selection always created a composite classifiers that were significantly more accurate than the component classifiers that they incorporated.

Component Diversity and Accuracy. In order to provide additional evidence that the four algorithms are actually doing something different, we now examine component diversity and accuracy together. Figures 4.9, 4.10, 4.11 and 4.12 graph the component accuracy against the component diversity for each of the four boosting algorithms and the 13 data sets. The graphs are grouped according to the general degree of success of boosting on that data. Figure 4.9 shows the four data sets where boosting performance was the worst, measured in terms of probability of significant improvement (p-values); Figure 4.10 shows the four data sets where boosting was the most successful; and Figures 4.11 and 4.12 show the remaining five data sets. Our goal is to show graphically that at least three algorithms effect different combinations of diversity and accuracy on almost every one of these data sets. As we have noted, Coarse Reclassification and Deliberate Misclassification typically show similar combinations of accuracy and diversity, according to the measures we have adopted.

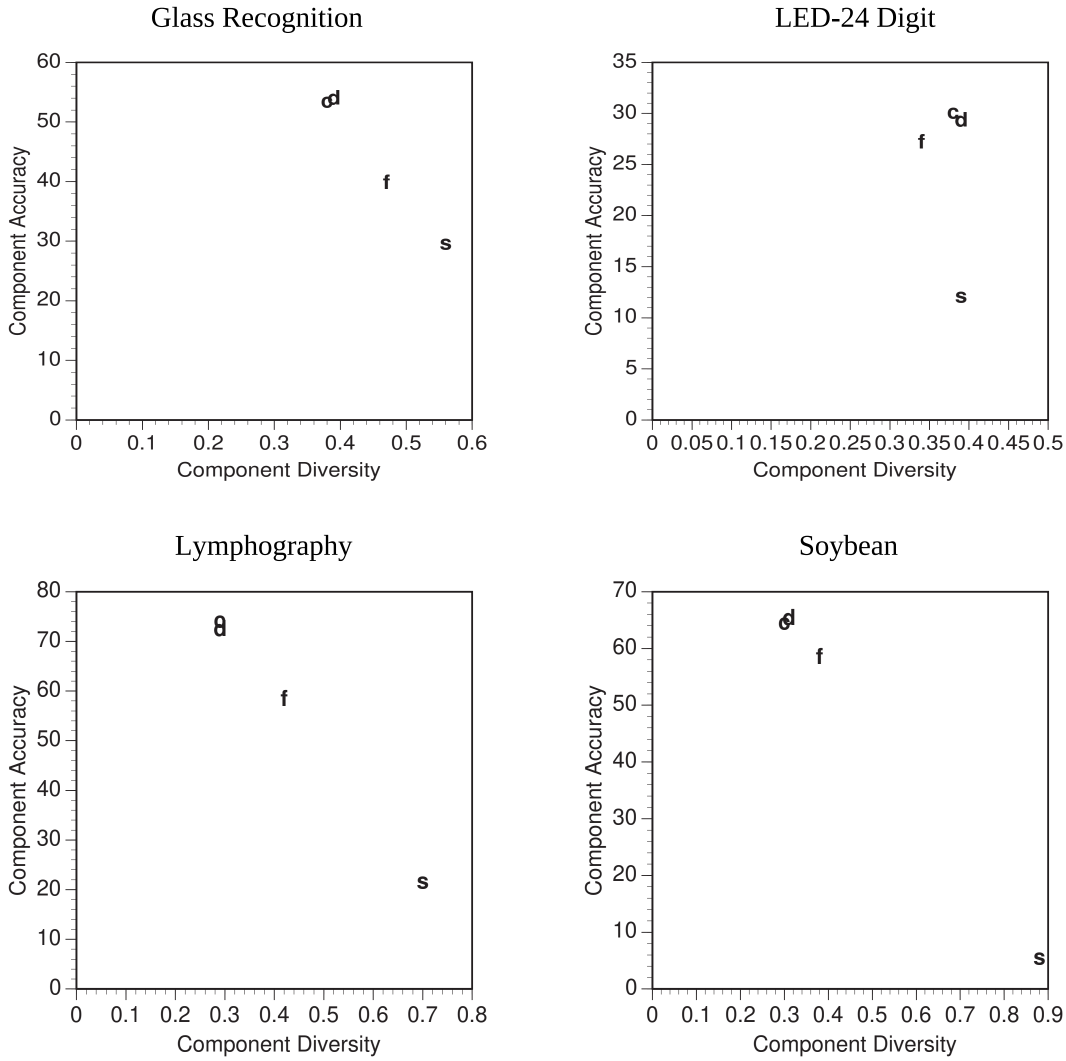

Figure 4.9: Relationships between component accuracy and diversity for the Glass Recognition, LED-24 Digit, Lymphography and Soybean data sets for the four boosting algorithms. "c" represents the Coarse Reclassification algorithm; "d", Deliberate Misclassification; "f", Composite Fitness; and "s" Composite Fitness–Feature Selection.

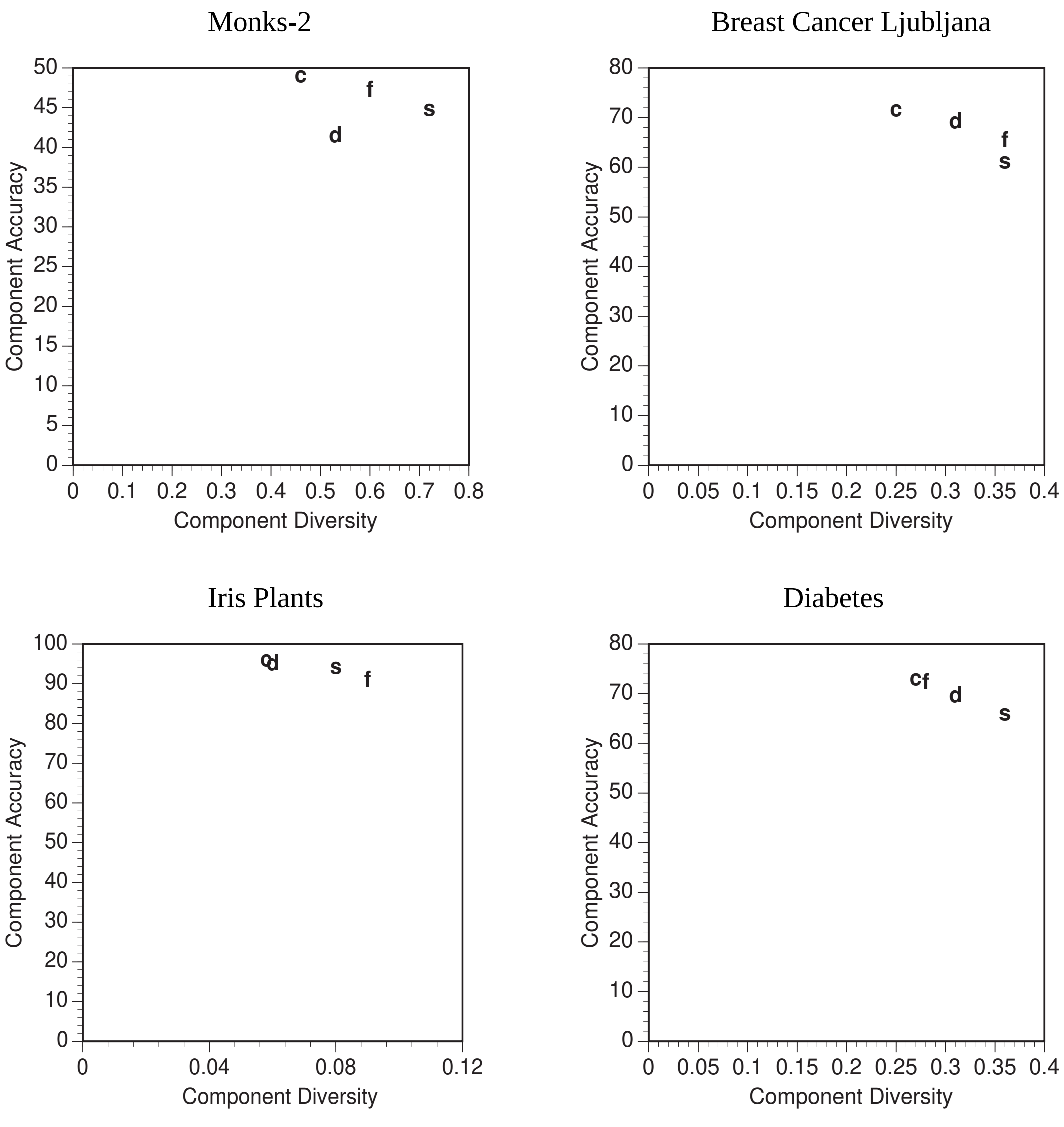

Figure 4.10: Relationships between component accuracy and diversity for the Monks-2, Breast Cancer Ljubljana, Diabetes and Iris Plants data sets for the four boosting algorithms. "c" represents the Coarse Reclassification algorithm; "d", Deliberate Misclassification; "f", Composite Fitness; and "s" Composite Fitness–Feature Selection.

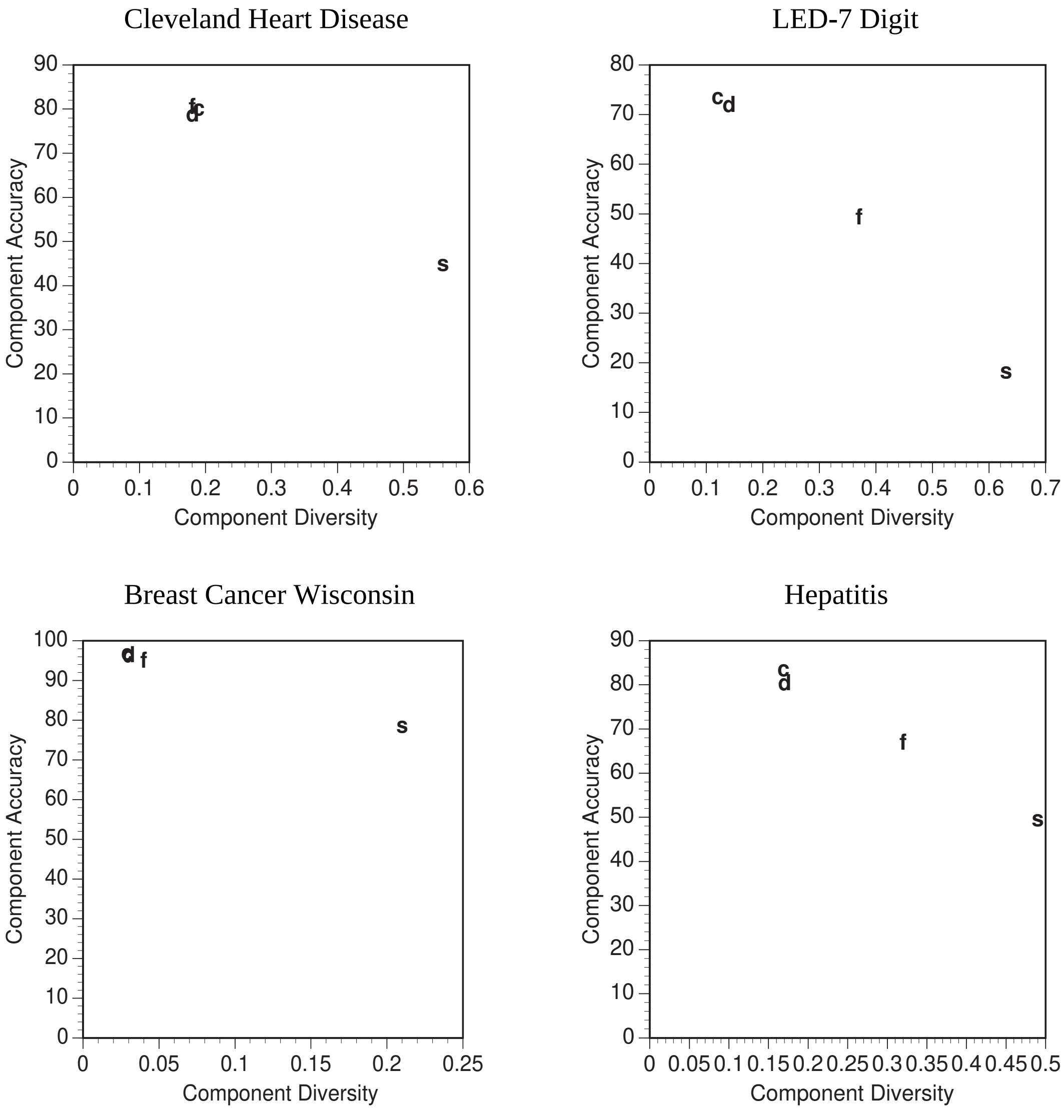

Figure 4.11: Relationships between component accuracy and diversity for the Cleveland Heart Disease, LED-7 Digit, Hepatitis and Breast Cancer Wisconsin data sets for the four boosting algorithms. "c" represents the Coarse Reclassification algorithm; "d", Deliberate Misclassification; "f", Composite Fitness; and "s" Composite Fitness–Feature Selection.

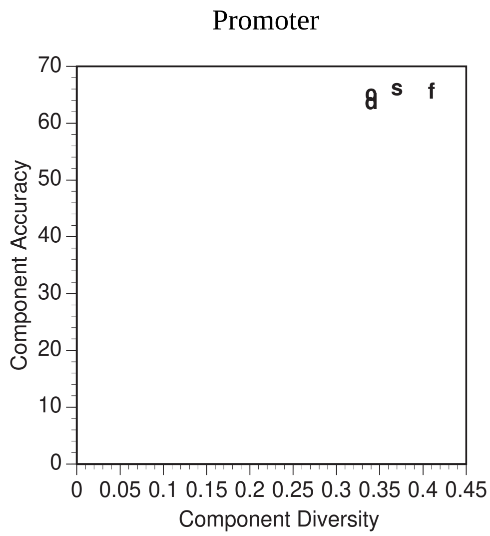

Figure 4.12: Relationships between component accuracy and diversity for the Promoter data set for the four boosting algorithms. "c" represents the Coarse Reclassification algorithm; "d", Deliberate Misclassification; "f", Composite Fitness; and "s" Composite Fitness–Feature Selection.

Again, we use Coarse Reclassification as a baseline algorithm and discuss the component diversity and accuracy on the other three.

Deliberate Misclassification. As we noted, Coarse Reclassification and Deliberate Misclassification generally show similar component accuracy and diversity. However, for three data sets, Deliberate Misclassification creates a component classifier with lower accuracy and higher diversity (e.g., Breast Cancer Ljubljana, Diabetes and Monks-2). These three data sets are all in the group where boosting was successful.

Composite Fitness. Coarse Reclassification typically shows higher component accuracy and lower diversity than Composite Fitness. Thus, where levels of component accuracy and diversity were implicitly selected according to their effect on composite training accuracy, the tradeoff was usually made in favor of selecting component classifiers that were less accurate and more diverse than the most accurate drawn from a sample.

On the other hand, there are three exceptions to this observation. Composite Fitness and Composite Fitness–Feature Selection attained approximately the same accuracy-diversity profile on the Diabetes and Cleveland Heart Disease data. Coarse Reclassification was more diverse and accurate than Composite Fitness on LED-24 Digit data, where both algorithms had substantially lower performance than Composite Fitness–Feature Selection. (Also, on the Promoter data, component accuracy was almost equal, but diversity was in fact higher.)

This observation shows that although composite accuracy for these two algorithms is often statistically indistinguishable (Table 4.12), boosting is achieved differently for Coarse Reclassification and Composite Fitness on almost all of the data sets tested. For the two exceptional data sets where accuracy and diversity were approximately equal, allowing the diversity and accuracy levels to be determined implicitly (as by Composite Fitness) yielded the same levels as simply trying to find the most accurate component (as by Coarse Reclassification).

Composite Fitness–Feature Selection. When we compare Composite Fitness–Feature Selection with Coarse Reclassification, we see that the Composite Fitness–Feature Selection components are almost always less accurate and more diverse than those created by Coarse Reclassification. For the Iris Plants data set, a one-feature component classifier was as accurate as a full nearest neighbor classifier. (This result confirms similar findings for the Iris Plants data [2].)

In this section, we have given evidence that at least three of four algorithms in general create component classifiers that show varying degrees of accuracy and diversity. The Deliberate Misclassification and Coarse Reclassification algorithms often show similar degrees of accuracy and diversity. The Composite Fitness and Composite Fitness–Feature Selection algorithms often created component classifiers that were more diverse and less accurate than the Coarse Reclassification algorithm. Since these algorithms often demonstrate comparable composite accuracy, the evidence gathered here suggests that boosting may not be a matter of finding a single, optimal accuracy-diversity tradeoff. In the next section, we continue the evaluation of the four boosting algorithms by comparing the algorithms to Schapire's original boosting algorithm [1990].

---

[2] Ron Kohavi, personal communication.

4.7 Comparison to Schapire's Boosting Algorithm

In this section, we ask, *How well do these algorithms work, relative to another boosting algorithm?* We use Schapire's original boosting algorithm as a benchmark. We refer to Schapire's algorithm as "Boosting" in this section. Boosting was described in Chapter 2. This algorithm is the closest boosting algorithm to our own, in that it attempts to combine only three component classifiers. The algorithm can be applied recursively, but ultimately the predictions according to only three hypotheses are combined [Schapire, 1990].

In brief, to improve the accuracy of a given classifier, Boosting constructs two additional component classifiers using a random training example selection method, and combines their predictions with the given classifier by voting [Schapire, 1990; Drucker *et al.*, 1994a]. In effect, Schapire's boosting algorithm constructs a composite classifier with three level-0 components whose predictions are combined by a level-1 voting algorithm. Since the four boosting algorithms add only one additional component classifier, the composite classifiers that we have been studying give rise to an even more concise level-1 representation.

Schapire's foundational algorithm was presented and analyzed within a theoretical, probably approximately correct (PAC)-learning framework in which an unlimited number of requests are permitted to be made for a random, labeled examples. The algorithm was proved correct only for two-class concepts and under this PAC-learning framework. In this framework, requests for examples may be made to an oracle until the learning algorithm generates a hypothesis that, with a certain high confidence, differs from the correct hypothesis with only a small error. Thus, for example, the algorithm says to "choose a sample sufficiently large that [a certain error bound is attained with a certain confidence]" [Schapire, 1990, p.203]. Further, under the PAC assumptions, the confidence and error must be attainable under any distribution of the examples.

Several of these assumptions pose difficulties for a practical implementation of Boosting. The assumption that an unlimited number of requests to an oracle is impractical. In a practical algorithm, we also have no way of assessing the confidence that a hypothesis is arbitrarily close to the true target concept. In the PAC framework, this confidence also determines how many calls to make to the example oracle. In general, Boosting cannot be implemented directly, since it is not clear how the PAC assumptions would be implemented.

To implement a version of Boosting, we assume that the base classifier, a full nearest neighbor classifier, stores all training instances. To limit the number of requests to an example oracle, we assumed that the second component classifier was trained on a training set that was at most half the size of the original training set, and computed according to the algorithm. The algorithm requires that the second component classifier be trained on a distribution of the instances on which the base classifier made and did not make errors. Choosing half the training set provided a midpoint between using all the training instances and using minimal prototype sets. Also, Breiman showed that training nearest neighbor component classifiers on bootstrap samples of training instances was not effective, and on average, bootstrap samples consist of approximately 63% of the individual training instances [1994a]. So we must choose a smaller number of instances in order to avoid nearest neighbor classifier stability. The third component classifier was then trained on all the training data on which the first two components disagree, to arbitrate between the predictions of the first two component classifiers, as the Boosting algorithm directs.

We compare two implementations of Boosting to the four boosting algorithms we have proposed. The two implementation differ in the model class that is used for the two classifiers that are used as components in addition to the base classifier. In the first implementation, *Boosting w/nn*, the two component classifiers added by Boosting are drawn from the model class of nearest neighbor classifiers that may incorporate any number of prototypes. In the second implementation, *Boosting w/minimal*, the two component classifiers are drawn from the model class of minimal nearest neighbor classifiers. In both implementations, the training instances are filtered according to Schapire's algorithm. The difference is that in *Boosting w/nn*, all the available training instances may be stored as prototypes; in *Boosting w/minimal*, the filtered training instances are used as the population from which minimal sets of prototypes are sampled. As in the prototype sampling algorithm described in Chapter 3, prototypes are drawn one from each class, and the prototype set with the highest training set accuracy is stored in a component classifier.

A comparison of *Boosting w/nn* and the baseline nearest neighbor algorithm is in Table 4.16.

Table 4.16: Accuracy of the *Boosting w/nearest neighbor* algorithm compared to nearest neighbor baseline.

Data	*NN*	*Boosting w/nn*	*Significance*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	69.3 ± 9.8	0.069
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	95.9 ± 1.5	0.828
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	77.0 ± 5.1	0.499
Diabetes $_{760}$	68.7 ± 5.1	67.1 ± 4.7	0.850
Glass Recognition $_{210}$	73.3 ± 10.3	72.9 ± 11.2	0.661
Hepatitis $_{150}$	77.3 ± 11.4	77.3 ± 11.0	0.500
Iris Plants $_{150}$	91.3 ± 7.1	91.3 ± 7.1	1.000
LED-24 Digit $_{200}$	36.5 ± 9.4	38.5 ± 9.7	0.187
LED-7 Digit $_{500}$	71.2 ± 7.7	71.6 ± 8.7	0.331
Lymphography $_{140}$	80.7 ± 8.9	80.0 ± 7.4	0.610
Monks-2 $_{430}$	48.8 ± 4.7	61.4 ± 7.5	0.000
Promoter $_{100}$	79.0 ± 9.9	81.0 ± 8.8	0.278
Soybean $_{300}$	91.0 ± 3.9	89.7 ± 5.1	0.888

In general, this implementation of Boosting did not boost nearest neighbor accuracy. The component classifiers store many instances, and therefore were fairly stable. This experiment is consistent with Breiman's results that stable nearest neighbor classifiers are not suitable for combination.

Boosting w/minimal fared somewhat better than *Boosting w/nn*, showing significant improvement over nearest neighbor on five data sets ($p \leq 0.05$).

The experimental results shown in Table 4.18 show that Coarse Reclassification significantly outperformed *Boosting w/minimal* on four data sets (Glass Recognition, Hepatitis, Monks-2 and Soybean) and Deliberate Misclassification outperformed it on three data sets (Glass Recognition, Monks-2 and Soybean). *Boosting w/minimal* was more accurate than Deliberate Misclassification on Diabetes.

Table 4.17: Accuracy of the *Boosting w/minimal* algorithm compared to nearest neighbor baseline.

Data	*NN*	*Boosting w/minimal*	*Significance*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	74.6 ± 6.4	0.002
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	97.1 ± 2.4	0.120
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	80.7 ± 5.6	0.038
Diabetes $_{760}$	68.7 ± 5.1	73.2 ± 3.1	0.002
Glass Recognition $_{210}$	73.3 ± 10.3	63.3 ± 6.7	0.994
Hepatitis $_{150}$	77.3 ± 11.4	81.3 ± 12.9	0.084
Iris Plants $_{150}$	91.3 ± 7.1	93.3 ± 7.0	0.140
LED-24 Digit $_{200}$	36.5 ± 9.4	33.5 ± 10.6	0.695
LED-7 Digit $_{500}$	71.2 ± 7.7	74.2 ± 7.8	0.013
Lymphography $_{140}$	80.7 ± 8.9	78.6 ± 14.3	0.740
Monks-2 $_{430}$	48.8 ± 4.7	55.1 ± 4.1	0.007
Promoter $_{100}$	79.0 ± 9.9	79.0 ± 11.0	0.500
Soybean $_{300}$	91.0 ± 3.9	81.3 ± 7.9	0.998

Table 4.18: Coarse Reclassification and Deliberate Misclassification compared to *Boosting w/minimal.*

Data	*Boosting w/minimal*	*CR*	*Sig. CR*	*DM*	*Sig. DM*
Breast Cancer Ljubljana $_{280}$	74.6 ± 6.4	73.1 ± 6.9	0.898	72.5 ± 6.9	0.888
Breast Cancer Wisconsin $_{690}$	97.1 ± 2.4	97.4 ± 1.9	0.256	97.3 ± 2.0	0.349
Cleveland Heart Disease $_{300}$	80.7 ± 5.6	79.5 ± 7.1	0.855	79.5 ± 4.4	0.888
Diabetes $_{760}$	73.2 ± 3.1	73.1 ± 5.1	0.517	71.6 ± 4.4	0.946
Glass Recognition $_{210}$	63.3 ± 6.8	70.3 ± 9.9	0.017	70.0 ± 10.0	0.024
Hepatitis $_{150}$	74.7 ± 10.3	80.2 ± 6.3	0.050	79.5 ± 7.4	0.094
Iris Plants $_{150}$	93.3 ± 7.0	95.1 ± 3.8	0.210	94.8 ± 3.3	0.209
LED-24 Digit $_{200}$	33.5 ± 10.6	27.2 ± 8.1	0.877	27.7 ± 9.4	0.849
LED-7 Digit $_{500}$	74.2 ± 7.8	73.6 ± 7.8	0.762	73.3 ± 7.3	0.884
Lymphography $_{140}$	78.6 ± 14.3	77.9 ± 10.7	0.590	76.7 ± 11.7	0.743
Monks-2 $_{430}$	55.1 ± 4.1	67.5 ± 5.8	0.000	65.8 ± 5.9	0.000
Promoter $_{100}$	79.0 ± 11.0	79.6 ± 7.0	0.434	80.7 ± 7.1	0.300
Soybean $_{300}$	81.3 ± 7.9	87.1 ± 4.4	0.024	87.9 ± 4.4	0.011

Table 4.19: Composite Fitness and Composite Fitness–Feature Selection compared to *Boosting w/minimal*.

Data	*Boosting w/minimal*	*CF*	*Sig. CF*	*CF-FS*	*Sig. CF-FS*
Breast Cancer Ljubljana $_{280}$	74.6 ± 6.4	75.2 ± 7.5	0.338	74.6 ± 8.8	0.500
Breast Cancer Wisconsin $_{690}$	97.1 ± 2.4	97.2 ± 1.8	0.363	96.7 ± 1.7	0.803
Cleveland Heart Disease $_{300}$	80.7 ± 5.6	80.1 ± 4.6	0.685	77.7 ± 4.5	0.953
Diabetes $_{760}$	73.2 ± 3.1	71.7 ± 4.6	0.912	73.7 ± 5.5	0.345
Glass Recognition $_{210}$	63.3 ± 6.8	70.5 ± 9.9	0.022	69.0 ± 7.5	0.022
Hepatitis $_{150}$	81.3 ± 12.9	78.7 ± 9.2	0.788	82.7 ± 10.5	0.015
Iris Plants $_{150}$	93.3 ± 7.0	94.0 ± 5.8	0.328	93.3 ± 5.4	0.500
LED-24 Digit $_{200}$	33.5 ± 10.6	26.8 ± 6.7	0.928	36.7 ± 10.0	0.258
LED-7 Digit $_{500}$	74.2 ± 7.8	72.7 ± 7.6	0.887	72.6 ± 8.1	0.916
Lymphography $_{140}$	78.6 ± 14.3	76.4 ± 10.6	0.739	76.7 ± 7.7	0.703
Monks-2 $_{430}$	55.1 ± 4.1	64.1 ± 7.3	0.002	61.9 ± 6.2	0.011
Promoter $_{100}$	79.0 ± 11.0	80.3 ± 6.7	0.383	82.5 ± 7.9	0.214
Soybean $_{300}$	81.3 ± 7.9	86.7 ± 4.9	0.010	87.7 ± 5.0	0.011

Table 4.19 shows that the Composite Fitness algorithm outperforms *Boosting w/minimal* on three data sets (Glass Recognition, Monks-2 and Soybean). *Boosting w/minimal* does better on the Diabetes and LED-24 Digit data sets. Composite Fitness–Feature Selection outperforms *Boosting w/minimal* on the Glass Recognition, Hepatitis, Monks-2 and Soybean data sets. *Boosting w/minimal* is superior on the Cleveland Heart Disease and LED-7 Digit data.

These comparisons show that the four boosting algorithms are significantly better than *Boosting w/minimal* on an average of 3.5 data sets. The four data sets on which *Boosting w/minimal* performed the worst are the same four on which our own algorithms fail. This observation may suggest that the success of boosting — using any algorithm — may depend largely on the component model classes involved. We devote the next section to explaining when the four boosting algorithms have failed to improve accuracy in our experiments and to summarizing when they have worked.

4.8 Analysis

4.8.1 Failure Analysis

In this section we ask, *When do these four algorithms fail?* The composite accuracies of the four algorithms are summarized in Table 4.20. These four algorithms have tended to perform poorly on four data sets: Glass Recognition, LED-24 Digit, Lymphography and Soybean, where they often yield a stacked generalizer that is less accurate than the base nearest neighbor classifier.

Table 4.20: Summary of the accuracy obtained by the four boosting algorithms.

Data	*NN*	*CR*	*DM*	*CF*	*CF-FS*
Breast Cancer Ljubljana $_{280}$	67.5	73.1	72.5	75.2	74.6
Breast Cancer Wisconsin $_{690}$	96.2	97.4	97.3	97.2	96.7
Cleveland Heart Disease $_{300}$	77.0	79.5	79.5	80.1	77.7
Diabetes $_{760}$	68.7	73.1	71.6	71.7	73.7
Glass Recognition $_{210}$	73.3	70.3	70.0	70.5	69.0
Hepatitis $_{150}$	77.3	80.2	79.5	78.7	82.7
Iris Plants $_{150}$	91.3	95.1	94.8	94.0	93.3
LED-24 Digit $_{200}$	36.5	27.2	27.7	26.8	36.7
LED-7 Digit $_{500}$	71.2	73.6	73.3	72.7	72.6
Lymphography $_{140}$	80.7	77.9	76.7	76.4	76.7
Monks-2 $_{430}$	48.8	67.5	65.8	64.1	61.9
Promoter $_{100}$	79.0	79.6	80.7	80.3	82.5
Soybean $_{300}$	91.0	87.1	87.9	86.7	87.7

As an initial matter, the mediocre or poor results on these data sets may be judged against Wolpert's observation that no generalizing scheme will always improve accuracy [1992, p.242]. Even important recent work in classifier combination fails on some data sets to show significant increases in accuracy [Brodley, 1994]. For example, the experiments of Ali and Pazzani [1996, p.128] reveal that some common U.C.I. data sets resist improvement through combination. For example, they show that combining rule-learning algorithms in eight different configurations never or rarely increased accuracy over a single invocation of the algorithm on the Breast Cancer Ljubljana, Diabetes, Hepatitis and Iris data sets. In 32 experiments with these data sets, their study showed a significant increase twice, which is within the realm of a random correlation at the 0.05 significance level. (At least one of the four boosting algorithms showed significant increases on all these data sets, incidentally, although improvements on Hepatitis were at a low level of confidence. Of course, we apply different baseline accuracies, component model classes and combining algorithms. Ali and Pazzani showed generally good results on Lymphography and Soybean, however, although their baseline accuracies were lower than for the full nearest neighbor classifier.)

Nevertheless, it will be useful to identify characteristics of the training data that are positively correlated with poor composite accuracy. One potential characteristic is that there may be too few instances per class. In this situation, sampling complementary classifier prototypes may suffer from a poverty of data in several ways. First, random stratified sampling more frequently will resample the same prototype(s) from a class. Second, where the number of instances per class is small, and those instances are isolated in the input space, none of the instances may accurately classify many members of that class.

Finally, where there are a large number of classes, each prototype may be responsible for determining several class boundaries (jointly with the other prototypes). The responsibility for determining several boundaries places strong constraints on a prototype, which may be difficult to satisfy simultaneously. Where the number of instances per class is low, it may be difficult to find a single prototype that satisfies these multiple constraints. These intuitions suggest that it may be difficult to select a complementary classifier that performs well as a level-0 component where there are few instances per class.

Table 4.21: Average number of training instances per class and increase observed from boosting. Boldface entries have the four lowest values for the mean number of training instances per class.

Data	*Training Instances*	*Classes*	*Mean Instances/Class*	*Increase*
Breast Cancer Ljubljana $_{280}$	252	2	126.0	7.7
Breast Cancer Wisconsin $_{690}$	621	2	310.5	0.5
Cleveland Heart Disease $_{300}$	270	2	135.0	3.1
Diabetes $_{760}$	684	2	342.0	5.0
Glass Recognition $_{210}$	189	6	**31.5**	-2.8
Hepatitis $_{150}$	135	2	67.5	5.4
Iris Plants $_{150}$	135	3	45.0	3.8
LED-24 Digit $_{200}$	180	10	**18.0**	0.2
LED-7 Digit $_{500}$	450	10	45.0	2.4
Lymphography $_{140}$	126	4	**31.5**	-2.8
Monks-2 $_{430}$	387	2	193.5	18.7
Promoter $_{100}$	90	2	45.0	3.5
Soybean $_{300}$	270	19	**14.2**	-3.1

Table 4.21 shows the relationship between the average number of instances per class and the increase derived from boosting. Column five of Table 4.21 shows the accuracy increase derived from boosting. To calculate the improvement, we took the best performing boosting algorithm (of the four) from Table 4.20 and subtracted the mean accuracy of the base nearest neighbor classifier. (So, for example, for Breast Cancer Ljubljana, the highest stacked generalizer accuracy is 75.2, the base nearest neighbor accuracy is 67.5, and so the difference is 7.7. It is apparently a coincidence that the mean instances/class and the improvement are equal for the Glass Recognition and Lymphography data.) This relationship between the mean instances per class (as independent variable) and the improvement through boosting, measured by this difference (as dependent variable) is graphed in Figure 4.13.

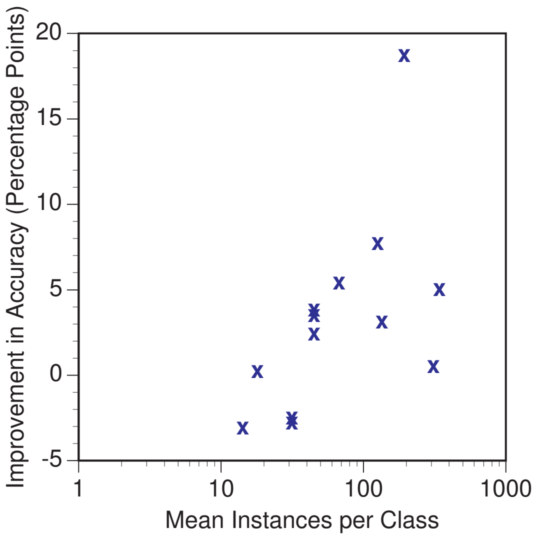

Figure 4.13: Relationship between mean instances per class and incremental improvement by boosting.

Figure 4.13 shows that our intuition is correct: the four data sets on which these algorithms failed to boost accuracy do in fact display the lowest mean number of training instances per class. These data sets (Glass Recognition, Lymphography, LED-24 Digit and Soybean) are represented by the four points toward the bottom left corner of the graph. (If we disregard the performance of Composite Fitness–Feature Selection on LED-24 Digit data, these are the four data sets for which "improvement" is strictly negative.)

The Breast Cancer Wisconsin data set shows that a small number of instances per class may be a sufficient condition, but is not a necessary condition, for low or negative boosting on these data sets, however. The improvement in accuracy for Breast Cancer Wisconsin is small (0.5%), although, within this collection, the number of training instances per class is large (310.5). However, one might expect a small degree of absolute improvement by boosting on data sets where the base nearest neighbor classifier is highly accurate (96.2%).

4.8.2 Selective Superiority

In this section we ask, *When do these boosting algorithms succeed?* The answer we give is simple: the boosting algorithms are selectively superior. On at least one data set, each of the four algorithms is superior to the other three.

Selective superiority refers to the inherent characteristic of learning algorithms that each performs well on some data sets, but that none is always the most accurate [Brodley, 1994]. Boosting algorithms are learning algorithms, and so it comes as no surprise that boosting algorithms are also selectively superior. One way to measure boosting algorithm superiority is the degree of statistical significance that its accuracy is higher than a baseline algorithm. For our four boosting algorithms, Table 4.22 lists the best algorithm under this measure.

Table 4.22: Selectively superior boosting algorithms, measured in terms of significance levels, which represent the level of significance for which the mean accuracy of the resulting stacked generalizer is greater than the mean accuracy of the full nearest neighbor classifier.

CR	**CF**
Breast Cancer Wisconsin $_{280}$ (0.075)	Cleveland Heart Disease $_{300}$ (0.042)
Lymphography $_{140}$ (0.860)	Glass Recognition $_{210}$ (0.443)
Monks-2 $_{430}$ (0.000)	Iris Plants $_{150}$ (0.009)
DM	**CF-FS**
Soybean $_{300}$ (0.998)	Breast Cancer Ljubljana $_{280}$ (0.005)
	Diabetes $_{760}$ (0.015)
	Hepatitis $_{150}$ (0.076)
	LED-7 Digit $_{500}$ (0.008)
	LED-24 Digit $_{200}$ (0.474)
	Promoter $_{100}$ (0.105)

A second way to determine superiority is in terms of maximal mean accuracy over multiple cross-validation runs. Table 4.23 lists the data sets for which each algorithm was best under that measure. As we demonstrated in Section 4.6, each algorithm may not be significantly better on the data sets listed; here, we appeal to nominal accuracy.

Table 4.23: Selectively superior boosting algorithms, measured in terms of maximal accuracy attained by the four boosting algorithms.

CR	**CF**
Breast Cancer Wisconsin $_{280}$ (96.7) Iris Plants $_{150}$ (95.1) LED-7 Digit $_{500}$ (73.6) Lymphography $_{140}$ (77.9) Monks-2 $_{430}$ (67.5)	Breast Cancer Ljubljana $_{280}$ (75.2) Cleveland Heart Disease $_{300}$ (80.1) Glass Recognition $_{210}$ (70.5)
DM	**CF-FS**
Soybean $_{300}$ (87.9)	Diabetes $_{760}$ (73.7) Hepatitis $_{150}$ (82.7) LED-24 Digit $_{200}$ (36.7) Promoter $_{100}$ (82.5)

Under both measures, each boosting algorithm is selectively superior on at least one data set. The algorithms are also selectively *inferior* in the following sense. On the four data sets that have proved the most resistant to boosting here (Glass Recognition, Lymphography, LED-24 Digit and Soybean), a different boosting algorithm showed the "best" performance (under both measures). Thus, it is impossible to make a simple conclusion of the form, "Boosting algorithm X is the most accurate of the four algorithms where low accuracy is obtained by all four." Future research may identify precisely when each algorithm should be applied.

4.9 Summary

In this chapter, we have explored boosting as a composite architecture to establish the hypothesis of this thesis: that the accuracy of a nearest neighbor classifier can be improved through composite-classifier architectures that incorporate a small number of diverse component nearest neighbor classifiers, each of which stores only a small number of prototypes. Our primary conclusion is that for a majority of these data sets, a single, minimal nearest neighbor classifier can be used as a component in a composite classifier to maintain or increase the accuracy of a base nearest neighbor classifier.

In the course of gathering evidence in favor of the hypothesis, we established a number of secondary results. These results were obtained by asking and answering several questions.

How well does each of the four algorithms boost the accuracy of a nearest neighbor classifier? Each of the four algorithms showed a statistically significant improvement over a nearest neighbor classifier on approximately half (six) of the data sets. On five of the 13 data sets, a significant ($p \leq 0.05$) improvement was shown by at least one of these algorithms, which add only one, minimal nearest neighbor component classifier for boosting. These results are notable in view of previously reported unsuccessful experiments to combine nearest neighbor classifiers to increase accuracy.

How well do these algorithms boost, when compared to each other? We used the performance of Coarse Reclassification as a baseline. On several data sets, at least one of the boosting algorithms was significantly more accurate than Coarse Reclassification. For the majority of data sets and algorithms, however, a statistically significant improvement over Coarse Reclassification was not evident.

How do these algorithms boost? Despite the similar accuracies gained by the four boosting algorithms, a difference was observed in the manner in which they boosted. The manner in which the algorithms boosted was characterized in terms of the accuracy and the diversity of the component classifiers. It was shown empirically that the relationship between accuracy and diversity is quite complex. Coarse Reclassification and Deliberate Misclassification frequently showed similar degrees of accuracy and diversity. The other three algorithms typically displayed different combinations of these two criteria. Where the degree of accuracy was allowed to be determined implicitly, as by the Composite Fitness algorithm, the components tended to display lower accuracy and greater diversity than the components created by Coarse Reclassification, where accuracy was the sole criterion. The component classifiers generated by the Composite Fitness–Feature Selection algorithm showed the lowest accuracy and the greatest diversity. These results suggest that there are different ways to achieve comparable composite accuracy, by varying the mix of component accuracy and diversity. Future research may be directed at developing families of boosting algorithms that automatically select the combination of accuracy and diversity for the data and the component model class.

How do these algorithms work, when compared to another boosting algorithm? Schapire's original boosting algorithm was used as the basis for comparison, since it employs only three component classifiers and it is therefore consistent with our design goals to use a small number of components [Schapire, 1990]. That algorithm was implemented in two ways. In the first implementation, the algorithm employed components selected from the model class of nearest neighbor classifiers that may employ any number of prototypes. That implementation did not perform well, and suffered from the same effects as observed by Breiman in bagging nearest neighbor classifiers. In the second implementation, the second and third component classifiers were chosen from the model class of minimal nearest neighbor classifiers. While the results were better than for the first implementation, the four boosting algorithms significantly outperformed the second implementation on an average of 3.5 data sets.

When do the four boosting algorithms fail? The four algorithms performed consistently poorly on four multi-class data sets (Glass Recognition, LED-24 Digit, Lymphography and Soybean). These four data sets display the smallest average number of instances per class of the data sets in our collection. We suggested several reasons why this characteristic may make it difficult to select a small number of prototypes to be used in a component classifier. The experiments provide preliminary evidence that minimal nearest neighbor classifiers are too coarse to be used to boost a full nearest neighbor classifier on these data sets.

When do they succeed? The four boosting algorithms were selectively superior on this data. Whether superiority was measured in terms of absolute accuracy or by the level of statistically significant improvement over a nearest neighbor classifier, one algorithm was superior to the other three on at least one data set.

CHAPTER 5

STACKED GENERALIZATION

We continue the investigation of incorporating minimal nearest neighbor classifiers as component classifiers in a composite classifier. In this chapter, we move from composite classifiers for boosting to stacked generalization. Stacked generalization gives a learning algorithm the flexibility to construct all the level-0 component classifiers, and they may be trained in parallel. In boosting, we have assumed that one of the level-0 components is fixed in advance. Boosting algorithms also typically train the components serially. Our goal in this chapter is to address a number of questions about the effects of varying several parameters that characterize a stacked generalizer. These characteristics are (a) the combining function, (b) the number of component classifiers, and (c) the number of prototypes in the component classifiers.

Conceptually, this chapter is organized into three parts. The first part compares three combining algorithms: ID3, nearest neighbor and voting (Section 5.1). The second part studies the effect of varying the number of component classifiers (Section 5.2). The third part investigates three additional issues: (1) the relationship among component diversity, component accuracy and composite accuracy, (2) class probability estimates for composite predictions and (3) the relationship between k-nearest neighbor classifiers and stacked generalizers (Sections 5.3, 5.4 and 5.5).

We treat these issues in response to a set of more-or-less natural questions. The first part of the chapter answers the question, *What effect does the level-1 combining function have on stacked generalization accuracy when a small number of minimal nearest neighbor classifiers are combined?* (Section 5.1). As we discussed in Chapter 2, voting is the default method for combining symbolic predictions, and so the question arises whether voting is as accurate as other computationally inexpensive and readily available learning algorithms. Although weighted voting is potentially useful in this context, we mean *equally-weighted* voting when we refer to *voting*. Specifically, where only minimal nearest neighbor level-0 classifiers are used, we want to know whether voting is to be preferred over other learning algorithms. We compare voting to a nearest neighbor algorithm and to ID3 as alternative combining algorithms.

The second part of the chapter begins with a question suggested by the boosting experiments in Chapter 4, where a minimal nearest neighbor classifier and a full nearest neighbor classifier were used as level-0 components. One next question is whether the full nearest neighbor classifier is necessary. In this second part, we ask, *Can a stacked generalizer incorporate only minimal nearest neighbor components and nevertheless yield accuracy higher than that of a full nearest neighbor classifier?* (Section 5.2). Trying to use majority vote to combine a number of weakly accurate components is also a "natural approach" [Schapire, 1990, p.198] to building highly accurate stacked generalizers. We discuss separately the usefulness of combining (only) two component classifiers (Section 5.2.1) and of combining more than two components (Section 5.2.2).

The third part of the chapter addresses additional questions. The first question returns to the fundamental issues of component diversity and component accuracy: *When minimal nearest neighbor classifiers are combined, what combination(s) of component accuracy and diversity will lead to high test set accuracy?* (Section 5.3).

The second question arises to address a possible extension to the composite classifiers created in this thesis. We have investigated stacked generalizers that output a class prediction only, but in many pattern recognition applications (e.g., zip-code recognition), it is desirable to have a measure of confidence in the prediction, for example, a (scalar) probability value. We therefore ask, *Is it possible to provide class probability estimates for stacked generalizers that incorporate minimal nearest neighbor components, where the components themselves output only a class label?* (Section 5.4).

In addition to addressing these questions, in the final section of this chapter we show that k-nearest neighbor classifiers can be viewed as stacked generalizers that combine a set of

" jth-nearest neighbor classifiers" (Section 5.5). We characterize a jth-nearest neighbor classifier as an instance-based classifier that outputs the class of the stored training instance that is the *jth-closest* neighbor according to a given distance metric. For example, a 3rd-nearest neighbor classifier predicts the class of the 3rd-closest stored instance. We use this generalization to create a variation of the k-nearest neighbor classifier that on three data sets shows significantly higher generalization accuracy than a k-nearest neighbor classifier.

5.1 Comparison of Combining Algorithms

The first question is whether different level-1 combining algorithms are appropriate for different data sets. The question may be viewed in terms of the selective superiority of learning algorithms [Brodley, 1994]. The selective superiority of learning algorithms entails that a given level-0 learning algorithm will be more accurate (relative to other learning algorithms) on some level-0 data than on other data, and that a level-1 learning algorithm will be more accurate on some level-1 data. The question we pose here is a slight variation on selective superiority: whether a given algorithm is superior as a level-1 combiner for certain *level-0* data sets.

This question of selective superiority "one level removed" is a difficult one because the level-0 component classifiers in effect transform the level-0 training data into level-1 training data. It is true that for any given set of level-0 component classifiers, this is a fixed transformation that should present only a minor complication. However, in constructing composite classifiers, the components are not given to us and must be constructed. The question of selective superiority "one level removed" is difficult because it entails determining the superiority of a level-1 learning algorithm for level-0 component classifiers from some model class(es), without knowledge of the particular component classifiers that are to be incorporated.

It is difficult to see how the question could be answered independently of the component classifiers and assumptions about them. However, in view of our design assumptions, a level-1 training set has at least two characteristics that distinguish that data from the data underlying many learning problems. These differences may affect the choice of combining algorithm.

First, there are typically only a small number of level-1 training instances. If there are n component classifiers and S is the set of class labels, the level-1 representation of an instance consists of members of the Cartesian product $(\prod_{i=1}^{n} S) \times S = \prod_{i=1}^{n+1} S$, comprised of one class label predicted by each of the n level-0 components, and the class of the (original) level-0 instance. We have assumed that the class labels are drawn from a finite set and that the number of labels is small. (Nineteen classes in the Soybean data is the greatest number of class labels in our collection; the median is two classes.) The number of level-1 instances is therefore at most $|S|^{n+1}$.

Second, ambiguous level-1 instances may be present; these are distinct instances where the vectors of component classifier predictions are equal, but the instance class labels are not equal. Where data from different classes are erroneously compressed into the same level-1 features, ambiguous instances will result. The number of unambiguous level-1 instances is therefore at most $|S|^{n}$.

These two characteristics make the level-1 data sets unlike many real-world data sets, including the real-world data sets that are found in the U.C.I. Repository [Murphy and Aha, 1994]. The question of the most accurate algorithm for combining is a specialized inquiry, which may not be answered by a comparison of algorithms on data with more general characteristics.

To compare combining algorithms on level-1 data, we performed two experiments. The first experiment compares the test accuracy of stacked generalizers that employ three combining algorithms — voting, nearest neighbor and ID3 — to combine *random* minimal nearest neighbor component classifiers. Although one (usually) would not set out to combine random classifiers, we use random classifiers to assess the performance of these combining algorithms in general. This approach may be reminiscent of early perceptron research, in which a perceptron incorporated a fixed, pre-processing layer of random functions [Rosenblatt, 1962; Sutton and Whitehead, 1993]. The second experiment compares the accuracy of stacked generalizers that use the three combining algorithms to combine the predictions of the minimal nearest neighbor components that are the *most accurate* in a sample of these classifiers. We cannot use the boosting algorithms of Chapter 4 for these experiments because only two component classifiers were used there. Using voting to combine two predictions would involve the resolution of ties that would have to be resolved according to some policy whenever the two classifiers disagreed.

Experiment 1. For each data set, we generated 100 sets of three random component nearest neighbor classifiers. We selected three components because three is the minimal number of classifiers to which voting realistically can be applied, and to observe our design decision to pursue efficiency by combining few components. (In Section 5.2.2 we investigate the combination of more components.) The component classifiers were randomly chosen by selectively sampling one prototype from each class. For each of the 100 sets of component classifiers, a stacked generalizer was created and trained with each of the three combining algorithms: voting, nearest neighbor and ID3. The nearest neighbor algorithm used an exact-match metric for the symbolic class labels and ID3 used the gain ratio feature selection metric.

Unlike C4.5 [Quinlan, 1993], ID3 is no longer considered a state-of-the-art decision tree algorithm. However, our implementation of ID3 uses the same feature selection metric as C4.5, gain ratio. Also, ID3 is applied to simple level-1 data. All the level-1 instance features are symbolic, there are no missing feature values, and there are very few features. Since the trees created on data of this sort are typically very small, pruning is not a major consideration, either. Thus, while C4.5 may be preferred in general to ID3, it is difficult to see how it would improve on ID3 for level-1 data with the characteristics encountered in these experiments.

We computed the mean test accuracy of the 100 stacked generalizers that incorporated each combining algorithm. This experiment was performed ten times, once for each cross-validation ply.

Analysis. Table 5.1 shows that, for all but two datasets, the nearest neighbor and ID3 algorithms yielded similar accuracy, but voting yielded lower accuracy than the other two algorithms. Although voting is a default method for combination of symbolic predictions, it performed very poorly with the random, low-accuracy minimal classifiers used in this

Table 5.1: Accuracies of stacked generalizers with random level-0 minimal nearest neighbor components, whose predictions are combined by three level-1 combining algorithms: voting, nearest neighbor and ID3. *Sig.NN-Vote* denotes the statistical significance that the mean accuracy of the stacked generalizers using nearest neighbor as the level-1 combining algorithm is higher than those that used voting. *Sig.ID3-Vote* and *Sig.ID3-NN* are defined analogously.

Data Set	*Vote*	*Sig. NN-Vote*	*NN*	*Sig. ID3-Vote*	*ID3*	*Sig. ID3-NN*
Breast Cancer Ljubljana $_{280}$	58.9	0.000	70.9	0.000	71.0	0.208
Breast Cancer Wisconsin $_{690}$	92.5	0.001	95.1	0.001	95.1	0.200
Cleveland Heart Disease $_{300}$	74.0	0.000	77.1	0.000	77.1	0.784
Diabetes $_{760}$	63.5	0.000	69.6	0.000	69.6	0.402
Glass Recognition $_{210}$	42.8	0.000	58.6	0.000	58.6	0.444
Hepatitis $_{150}$	70.2	0.001	79.3	0.001	79.2	0.701
Iris Plants $_{150}$	87.8	0.000	89.2	0.000	89.3	0.315
LED-24 Digit $_{200}$	29.2	0.914	27.6	0.997	26.9	0.872
LED-7 Digit $_{500}$	60.1	0.000	70.9	0.000	71.0	0.376
Lymphography $_{140}$	62.2	0.000	69.2	0.000	68.9	0.777
Monks-2 $_{430}$	43.9	0.000	66.4	0.000	66.5	0.018
Promoter $_{100}$	66.8	0.880	65.9	0.961	65.4	0.890
Soybean $_{300}$	62.4	0.000	72.6	0.000	72.6	0.514

experiment. In general, this experiment therefore suggests that voting should not be applied to a small number of classifiers that have low accuracy.

The exceptions to the general pattern were the LED-24 and Promoter data sets, where voting was superior to the other two algorithms. The differences in accuracy among the three algorithms were small, however. Further, on these two data sets, the stacked generalizer applying any one of the level-1 learning algorithms showed much lower accuracy than the baseline nearest neighbor classifier. In our collection, these two data sets are those (a) that have solely binary or symbolic features and also (b) have the fewest number of instances per feature (computed from Table 3.1): in increasing order, these are Promoter, 1.8 instances/feature; LED-24 Digit, 8.3; Soybean, 8.8; LED-7 Digit, 71.4; and Monks-2, 72.0. In this experiment, voting was a more accurate algorithm for combining randomly selected minimal nearest neighbor classifiers when the original level-0 instances are drawn from a sparse, discrete, high-dimensional space. On the other hand, more research is needed to establish this hypothesis, since our sample size of 13 data points, one for each data set in the collection, is so small. Even within this small collection, the Soybean data set has nearly the same number of instances per feature as LED-24 Digit data (8.8 vs. 8.3), but for Soybean data, voting was less accurate than nearest neighbor and ID3.

Experiment 2. The second experiment compared the three combining algorithms where the component classifiers were more accurate on the training set than those chosen at random. One hundred minimal nearest neighbor classifiers were sampled, by sampling one prototype from each class. Of these 100 classifiers, the three nearest neighbor classifiers that showed the highest accuracies on the training set were selected as components.

Table 5.2 gives the mean accuracy of the stacked generalizers that incorporate those three most accurate classifiers as components and that apply each of the three learning algorithms as the level-1 combining method.

Table 5.2: Accuracies of stacked generalizers with the three level-0 minimal nearest neighbor components with the highest training accuracy, whose predictions are combined by three level-1 combining algorithms: voting, nearest neighbor and ID3.

Data Set	*Vote*	*Sig. NN-Vote*	*NN*	*Sig. ID3-Vote*	*ID3*	*Sig. ID3-NN*
Breast Cancer Ljubljana $_{280}$	73.2	0.041	74.3	0.097	74.3	0.500
Breast Cancer Wisconsin $_{690}$	96.7	0.339	96.8	0.339	96.8	0.500
Cleveland Heart Disease $_{300}$	79.7	0.952	78.0	0.948	78.3	0.172
Diabetes $_{760}$	74.3	0.222	74.6	0.222	74.6	0.500
Glass Recognition $_{210}$	53.8	0.000	64.3	0.000	66.2	0.052
Hepatitis $_{150}$	80.7	0.172	82.0	0.172	82.0	0.500
Iris Plants $_{150}$	96.7	0.500	96.7	0.500	96.7	0.500
LED-24 Digit $_{200}$	35.0	0.980	24.5	0.994	20.5	0.965
LED-7 Digit $_{500}$	73.0	0.009	75.2	0.009	75.6	0.084
Lymphography $_{140}$	75.0	0.740	72.9	0.761	72.1	0.704
Monks-2 $_{430}$	48.1	0.000	64.2	0.000	64.2	0.500
Promoter $_{100}$	67.0	0.056	73.0	0.043	75.0	0.172
Soybean $_{300}$	68.3	0.151	71.3	0.402	69.0	0.914

Analysis. In this experiment, the component classifiers were more accurate than random classifiers, and there was a smaller difference between the accuracy of voting and the other two algorithms. At the $p \leq 0.05$ level of significance, for 9 of 13 data sets there was no significant improvement by nearest neighbor or ID3 over voting. Voting was the best-performing combiner on LED-24 Digit data here and in the previous experiment. These two experiments, where voting was a superior combiner for LED-24 Digit data, suggest the hypothesis that voting is a better combiner where the data contain many noisy and irrelevant features. (LED-7 Digit, where voting was less accurate, also has feature noise, but has no irrelevant features.) In any event, more than 13 data points would be needed to establish this hypothesis as to the utility of voting any certain circumstances.

ID3 and nearest neighbor perform comparably, except that nearest neighbor was more accurate on the LED-24 data, and ID3 was more accurate on the Glass Recognition data.

Together, these two experiments show that the choice of level-1 learning algorithm depends on the level-0 data and also on the accuracy of the level-0 component classifiers. Voting is less accurate for random components and slightly less accurate on components that are the most accurate in a sample of minimal nearest neighbor classifiers.

5.2 Number of Component Classifiers

In the previous section, we investigated the effect of the level-1 algorithm on the combination of minimal nearest neighbor components. In this section, we return to a focus on the level-0 component classifiers, where we show the effects of combining a number of minimal component classifiers. In the course of these experiments, we also show the results of combining an equal number of nearest neighbor classifiers that contain many prototypes, rather than one per class.

In this section, we show the results of combining two, three and more than three component classifiers. We investigate three components separately because a composite classifier with three components is the *minimal* stacked architecture to which all three combining algorithms (nearest neighbor, ID3 and voting) reasonably can be applied, and because minimal architectures are the focus of this thesis. In this section, we use ID3 as a combining algorithm where two components are present; we use voting where more than two components are present.

Table 5.3 summarizes the experiments that we perform in this section. The experiments may be characterized in terms of the number of component classifiers, the combining function and the number of prototypes stored in each component (Table 5.3). By varying these dimensions, we explore a larger part of the search space for nearest neighbor classifier combination. Bagging nearest neighbor classifiers, which involves combining a large number of classifiers that store large numbers of prototypes, has been investigated by Breiman [1994a] and we do not repeat his experiments here.

Table 5.3: Sections and experiments on combining nearest neighbor classifiers.

	Number of Components and Combiner		
	ID3	*Voting*	
Prototype Set	*2*	*3*	*> 3*
Minimal	Sec. 5.2.1 (Exp. 2)	Sec. 5.2.2 (Exp. 2)	Sec. 5.2.2 (Exp. 2)
Larger	Sec. 5.2.1 (Exp. 1)	Sec. 5.2.2 (Exp. 1)	bagging

5.2.1 Two Component Classifiers

Our experiments with boosting show that, on a variety of data sets, the accuracy of a full nearest neighbor classifier can be improved by combining it with a minimal nearest neighbor classifier. In this section, we ask whether the full nearest neighbor classifier is a *necessary* component: is it possible to combine *two minimal* classifiers to achieve accuracy higher than a classifier that incorporates all instances as prototypes? We try to answer this question experimentally.

As a preliminary experiment, we show that combining two nearest neighbor classifiers that store a large number of prototypes will not improve classification accuracy significantly. This experiment is a variant of Breiman's demonstration of the inability of bagging to improve nearest neighbor accuracy [Breiman, 1994a]. Although bagging uses voting to combine the component predictions, we use ID3, because only two components are present.

In these experiments that use a decision tree level-1 algorithm to combine the predictions of a set of nearest neighbor component classifiers, we compare the accuracy of the resulting stacked generalizer with that of C4.5, used as a standalone classifier. C4.5 is a state-of-the-art decision-tree algorithm [Quinlan, 1993] that uses the same feature selection metric that is applied by ID3 in these experiments (gain ratio). C4.5 was run with the default options specified by Quinlan [1993]. The accuracy of pruned trees was used for comparison. Pruned trees outperformed unpruned trees on 9 of the 13 data sets.

Experiment 1 (Two components, bootstrap samples of prototypes). In this experiment, we combined two nearest neighbor classifiers using ID3 as a combining classifier. Each nearest neighbor classifier stored a bootstrap sample of prototypes. Recall that a bootstrap sample is a sample taken with replacement [Breiman, 1994a]. On average, approximately 63.2% of a population of instances will appear (one or more times) in a bootstrap sample whose size is the number of elements in the population [Breiman, 1994a]. Here, the sample size was the number of instances in the training set. Ten-fold cross validation was performed and mean stacked generalizer test accuracy is reported in Table 5.4.

Table 5.4: Mean test accuracies of stacked generalizers (SG) that use ID3 to combine two nearest neighbor classifiers that store bootstrap samples of prototypes.

Data	*NN*	*C4.5*	*SG*	*Sig. NN*	*Sig. C4.5*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	72.5 ± 6.5	68.9 ± 7.5	0.348	0.941
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	95.1 ± 2.7	96.5 ± 1.8	0.172	0.062
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	76.3 ± 9.6	77.3 ± 6.6	0.416	0.363
Diabetes $_{760}$	68.7 ± 5.1	71.3 ± 5.5	67.9 ± 3.9	0.725	0.995
Glass Recognition $_{210}$	73.3 ± 10.3	68.1 ± 9.0	71.0 ± 7.3	0.794	0.542
Hepatitis $_{150}$	77.3 ± 11.4	81.4 ± 9.8	74.7 ± 12.5	0.865	0.963
Iris Plants $_{150}$	91.3 ± 7.1	96.7 ± 4.7	92.0 ± 6.9	0.172	0.988
LED-24 Digit $_{200}$	36.5 ± 9.4	70.0 ± 16.2	28.0 ± 11.8	0.987	0.999
LED-7 Digit $_{500}$	71.2 ± 7.7	73.6 ± 8.2	69.8 ± 8.5	0.779	0.972
Lymphography $_{140}$	80.7 ± 8.9	77.8 ± 10.4	78.6 ± 8.9	0.783	0.410
Monks-2 $_{430}$	48.8 ± 4.7	67.0 ± 7.4	67.0 ± 7.4	0.000	0.633
Promoter $_{100}$	79.0 ± 9.9	76.0 ± 14.3	85.0 ± 11.8	0.084	0.034
Soybean $_{300}$	91.0 ± 3.9	86.3 ± 5.3	85.0 ± 6.9	0.999	0.685

Analysis. Significantly better results than nearest neighbor were attained only on the Monks-2 data. Significantly better results than C4.5 were attained on the Promoter data. A t-test reveals that on three data sets (Diabetes, Iris and LED-24 Digit), C4.5 is significantly ($p \leq 0.05$) more accurate than the stacked generalizers created in this experiment. This experiment is consistent with Breiman's observed failure to improve accuracy by bagging nearest neighbor classifiers.

In the next experiment, we show that combining two minimal component classifiers can lead to higher accuracy on more than just one data set, although poor results are also attained on at least five data sets.

Experiment 2 (Two components, minimal prototype sets). One hundred minimal nearest neighbor classifiers were randomly created, by sampling one prototype from each class, as usual. The two most accurate classifiers were used as the component classifiers in a stacked generalizer. The combining algorithm used was ID3. The stacked generalizer was trained on a training set and the test accuracy was recorded. The experiment was performed 20 times, on two identical 10-fold cross validation partitions. The mean accuracy is given.

Table 5.5: Test accuracy of a stacked generalizer (SG) that incorporates two minimal nearest neighbor components and ID3 as the combining algorithm.

Data	*NN*	*C4.5*	*SG*	*Sig. NN*	*Sig. C4.5*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	72.5 ± 6.5	72.3 ± 8.8	0.074	0.551
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	95.1 ± 2.7	96.7 ± 2.2	0.311	0.028
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	76.3 ± 9.6	81.2 ± 4.9	0.023	0.045
Diabetes $_{760}$	68.7 ± 5.1	71.3 ± 5.5	75.2 ± 4.0	0.002	0.000
Glass Recognition $_{210}$	73.3 ± 10.3	68.1 ± 9.0	64.0 ± 9.7	0.963	0.857
Hepatitis $_{150}$	77.3 ± 11.4	81.4 ± 9.8	81.7 ± 9.2	0.098	0.435
Iris Plants $_{150}$	91.3 ± 7.1	96.7 ± 4.7	96.0 ± 5.4	0.015	0.776
LED-24 Digit $_{200}$	36.5 ± 9.4	70.0 ± 16.2	20.7 ± 5.3	0.999	1.000
LED-7 Digit $_{500}$	71.2 ± 7.7	73.6 ± 8.2	74.5 ± 7.6	0.023	0.240
Lymphography $_{140}$	80.7 ± 8.9	77.8 ± 10.4	75.4 ± 8.3	0.967	0.827
Monks-2 $_{430}$	48.8 ± 4.7	67.0 ± 7.4	67.0 ± 7.4	0.000	0.632
Promoter $_{100}$	79.0 ± 9.9	76.0 ± 14.3	66.5 ± 7.5	0.990	0.973
Soybean $_{300}$	91.0 ± 3.9	86.3 ± 5.3	70.7 ± 6.0	1.000	1.000

Analysis. Table 5.5 shows that combining two minimal nearest neighbor classifiers yields stacked generalizers that are either quite accurate or quite inaccurate. With respect to our specified nearest neighbor baseline, for all but one data set (Breast Cancer Wisconsin), the significance level was either less than 0.10 or greater than 0.95. On the four data sets where boosting was found to be difficult in Chapter 4, (Glass Recognition, LED-24 Digit, Lymphography and Soybean) and on the Promoter data, the accuracy of these stacked generalizers was also worse than a nearest neighbor classifier. As we have observed, the LED-24 Digit, Promoter and Soybean data sets have binary or symbolic values and the fewest instances per feature in this collection. Lymphography also has a large number of symbolic features: 15 of 18 are symbolic. The exact-match distance metric used by the component classifiers for symbolic values is not very sophisticated and may contribute to the poor results for symbolic data in sparse, high-dimensional spaces.

On the other hand, Glass Recognition data set has nine real-valued features and no symbolic ones. In Chapter 3, it was shown that Glass Recognition is one data set where a larger number of prototype samples would have improved accuracy.

On five data sets, the stacked generalizers in this experiment showed significantly higher accuracy ($p \leq 0.05$) than a nearest neighbor classifier. Further, on six data sets, the stacked generalizers in this experiment showed improvement over a nearest neighbor algorithm at a higher level of confidence (i.e., a lower level of statistical significance) than all of the Chapter 4 boosting algorithms (Cleveland Heart Disease, Diabetes, Hepatitis, Iris Plants, LED-7 Digit and Lymphography).

With respect to C4.5, the stacked generalizers created in this experiment were significantly more accurate than C4.5 on three data sets (Breast Cancer Wisconsin, Cleveland Heart Disease and Diabetes). A t-test shows that C4.5 was significantly more accurate than the stacked generalizers on three data sets (LED-24 Digit, Promoter and Soybean).

A t-test also showed that these composite classifiers were not significantly more accurate than their components, in general. We can quickly summarize the results of this test: only on the Glass Recognition and Monks-2 data sets were the composite classifiers significantly more accurate on average than both the components ($p \leq 0.05$) on both of the two 10-fold cross validation runs. Composite accuracy on Glass Recognition and Promoter was poor, however. In summary, combining only two minimal component classifiers leads to mixed results on this collection.

5.2.2 More than Two Component Classifiers

Two features of our design plan for composite classification have been to use few component classifiers and to use component classifiers that are inexpensive to apply. In this section, we study the effects of relaxing these constraints.

We perform two experiments in this section. Although the experiments are designed to show different effects, they are similar in that they use voting as the combining algorithm.

The first experiment is designed to investigate the combination of three nearest neighbor classifiers that store many prototypes. In Section 5.2.1, we showed that combining two nearest neighbor classifiers that store bootstrap samples of prototypes does not improve accuracy in general. In this section, we increase the number of components to three. With three components, voting may be used to combine the predictions.

In a set of bagging experiments, Breiman has shown on six data sets that combining a large number of nearest neighbor classifiers (100) that store bootstrap samples of prototypes does not improve accuracy over a nearest neighbor algorithm [Breiman, 1994a]. Bagging uses voting to combine symbolic predictions of the component classifiers, and we follow that lead for the experiments in this section. Also, we showed in Section 5.1 that voting was an effective combiner for accurate components.

In the second experiment, we investigate the effect on composite accuracy of relaxing the constraint that the number of component minimal classifiers be very small. We show that, on as many as six data sets in our collection, combining three or more minimal component classifiers by voting yields significantly higher accuracy than a full nearest neighbor classifier. Very poor results are seen on four data sets, however.

Experiment 1 (Three components, bootstrap samples of prototypes). To determine the effectiveness of combining a small number of nearest neighbor classifiers that store many prototypes, we constructed three nearest neighbor classifiers to be combined by voting. For each classifier, a bootstrap sample was taken of the training instances, where the sample size was the number of instances in the training set. Ten-fold cross validation was performed.

Analysis. Table 5.6 shows significant improvement ($p \leq 0.05$) on one data set (Monks-2). These results complement Breiman's results by showing that a bootstrap sample of prototypes does not typically improve accuracy over a nearest neighbor classifier. On the three data sets that our experiments have in common with those reported by Breiman (Breast Cancer Wisconsin, Diabetes [1] and Glass Recognition), these experiments confirm the results reported by Breiman in that no significant increase was seen over nearest neighbor accuracy.

Table 5.6: Bagging three nearest neighbor components.

Data	*NN*	*Bagging*	*Significance*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	68.6 ± 9.3	0.097
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	96.1 ± 1.2	0.661
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	77.0 ± 4.8	0.500
Diabetes $_{760}$	68.7 ± 5.1	69.9 ± 5.1	0.234
Glass Recognition $_{210}$	73.3 ± 10.3	72.4 ± 11.8	0.778
Hepatitis $_{150}$	77.3 ± 11.4	74.7 ± 8.8	0.813
Iris Plants $_{150}$	91.3 ± 7.1	94.0 ± 4.9	0.052
LED-24 Digit $_{200}$	36.5 ± 9.4	39.5 ± 10.1	0.120
LED-7 Digit $_{500}$	71.2 ± 7.7	67.0 ± 8.9	0.963
Lymphography $_{140}$	80.7 ± 8.9	81.4 ± 7.7	0.390
Monks-2 $_{430}$	48.8 ± 4.7	56.3 ± 7.1	0.008
Promoter $_{100}$	79.0 ± 9.9	83.0 ± 14.2	0.084
Soybean $_{300}$	91.0 ± 3.9	88.3 ± 6.1	0.931

The second experiment investigates the effects of combining a larger set of unstable nearest neighbor components, which individually incorporate only a small number of prototypes. In this experiment, voting is also used as the combining algorithm to investigate what Schapire [1990] calls the "natural approach" to stacking: combining a set of weak classifiers by voting. Kearns has shown by counterexample that this approach cannot be provably correct [1988].

Experiment 2 (Three and more than three components, minimal prototype sets). One hundred samples of prototypes were taken, one per class, and stored in nearest neighbor

[1] In the experiments reported by Professor Breiman, a variation of the Diabetes was used. The variation equalized the class sizes by duplicating instances.

classifiers to classify the training set. These minimal classifiers were ranked in decreasing order of training accuracy, from the most accurate to the least accurate. In this experiment, we used as components the n most accurate classifiers in this ranking, for $n = 3, 5, 11, 21$. Voting was used to combine the predictions. An odd number of components was used in each case to avoid ties in voting. In Table 5.7 we show the mean test accuracy of the 3, 5, 11 and 21-component stacked generalizers.

Analysis. The stacked generalizers showed significant improvement over a full nearest neighbor classifier at the $p \leq 0.05$ significance level or better on two data sets (three components), five data sets (five components), and six data sets (11 and 21 components). These results show that on almost half the data sets in this collection, a moderate number of minimal nearest neighbor classifiers can be combined to increase accuracy significantly over a full nearest neighbor classifier.

Poor results were seen on at least three data sets (11 components) or four data sets (3, 5 and 21 components). Where voting was less accurate than the baseline nearest neighbor classifier, typically it was substantially much less accurate (Glass Recognition, Lymphography, Promoter and Soybean data).

In Table 5.7 we have also flagged (with boldface) the entries corresponding to stacked generalizers that used components whose median accuracy was not significantly higher than the baseline nearest neighbor algorithm. These stacked generalizers exhibited accuracy significantly *higher* than the nearest neighbor algorithm by combining components whose accuracy was *lower than or comparable to* a nearest neighbor classifier.

We appeal to median component accuracy in this analysis in view of voting's behavior as combiner. For two-class data, the median accuracy provides a threshold for voting. In the two-class problems (see Table 3.1 for a summary of the data sets), where more than half of the component classifiers correctly classified an instance, the stacked generalizer was also correct. Of course, when fewer than half of the components were correct, the stacked generalizer was incorrect. For multi-class data sets, the median accuracy is a convenient measure of the aggregate accuracy of the components. Where voting is the combiner, maximum accuracy is a less useful measure of component accuracy, because the most accurate classifier can be outvoted, sometimes easily.

As more components were used (11 and 21), the median classifier was less accurate, and all of the stacked generalizers that improved significantly on the nearest neighbor algorithm used components that were not significantly more accurate than nearest neighbor.

We can also observe the pattern of accuracy as more components are combined. Monotonic improvements were seen on the LED-24 Digit, LED-7 Digit and Promoter data sets as the number of classifiers was increased from 3 to 21. The LED-24 Digit recognition data set has 17 irrelevant features and 10% feature value noise. The accuracies achieved on this data provide evidence that combining minimal nearest neighbor components can help overcome the presence of irrelevant features and feature value noise. For other data sets, such as Glass Recognition and Monks-2, as more classifiers are combined, the predictions of the less accurate classifiers tended to dominate predictions and thereby lower the composite accuracy.

Stacking minimal nearest neighbor classifiers also results in a classifier that is at least as efficient to apply as a nearest neighbor classifier. A smaller number of distinct prototypes are stored in the level-0 classifiers in the aggregate than are stored by a full nearest neighbor classifier, since some training instances may not be stored by any component. Where instances are stored as prototypes by more than one component, the calculation of the distance from a test instance to each such prototype may be computed once and then cached for retrieval by other components. The additional computational cost is small for combining the component predictions by voting. Thus, where the voting stacked generalizer has statistically comparable or higher accuracy (*e.g.,* Breast Cancer Ljubljana, Breast Cancer Wisconsin, Cleveland Heart Disease, Diabetes, Hepatitis, Iris Plants, LED-24 Digit and LED-7 Digit), it can be more efficient to apply than a full nearest neighbor classifier and at least as accurate.

In summary, in this section we have explored a larger part of the search space of stacked generalizers that use nearest neighbor classifier components. In general, combining classifiers that stored a bootstrap sample of prototypes did not improve accuracy over a nearest neighbor classifier. Combining minimal nearest neighbor classifiers showed improved accuracy on up

Table 5.7: Test accuracy of stacked generalizers that combine 3, 5, 11 and 21 minimal nearest neighbor component classifiers by voting. A "*" indicates significant improvement over the nearest neighbor classifier at the 0.05 significance level. A "†" indicates significantly worse accuracy at that level, where a t-test shows that the nearest neighbor classifier has significantly higher accuracy than the composite. Boldface entries indicate stacked generalizers that display significantly higher accuracy than nearest neighbor and whose median component accuracy is not significantly higher than the baseline nearest neighbor classifier.

		Components			
Data	*NN*	*3*	*5*	*11*	*21*
Breast Cancer Ljubljana $_{280}$	67.5	71.8	73.6*	72.5	**73.2***
Breast Cancer Wisconsin $_{690}$	96.2	96.2	96.7	96.2	95.9
Cleveland Heart Disease $_{300}$	77.0	80.0	80.3	**82.0***	**81.3***
Diabetes $_{760}$	68.7	73.3*	74.7*	**73.4***	72.6
Glass Recognition $_{210}$	73.3	62.4†	58.6†	54.3†	54.8†
Hepatitis $_{150}$	77.3	**84.0***	**82.0***	**82.7***	**83.3***
Iris Plants $_{150}$	91.3	94.7	95.3*	**96.7***	**94.7***
LED-24 Digit $_{200}$	36.5	33.0	36.5	**43.5***	**50.0***
LED-7 Digit $_{500}$	71.2	73.2	**74.4***	**75.0***	**75.2***
Lymphography $_{140}$	80.7	71.4†	73.6†	79.3	75.0
Monks-2 $_{430}$	48.8	48.6	45.6	44.9	41.2†
Promoter $_{100}$	79.0	63.0†	64.0†	67.0†	70.0†
Soybean $_{300}$	91.0	71.0†	71.7†	71.3†	76.3†

to six data sets. Poor accuracy was typically seen on at least four data sets. We summarize the experiments in this section by counting the number of data sets on which significant improvements through combining were observed (Table 5.8).

Table 5.8: Summary of experiments in this section: number of data sets on which significant improvement over a nearest neighbor algorithm was shown.

	Number of Components and Combiner		
	ID3	*Voting*	
Prototype Set	*2*	*3*	> *3*
Minimal	5	2	≤ 6
Bootstrap	1	1	[bagging: 0]

The bagging result is reported by Breiman [1994a] on a different collection of data sets. The results in this section provide additional evidence of the lack of utility in combining nearest neighbor classifiers that store many prototypes and suggest that our policy of *de-stabilizing* nearest neighbor classifiers is useful to achieve improvement on some data sets through nearest neighbor classifier combination.

5.3 Relationship between Accuracy and Diversity

While it has been generally recognized that component classifiers should be accurate and diverse [Ali and Pazzani, 1996], the relationship between accuracy and diversity has not been characterized analytically or empirically. In one algorithm where a relationship was explicit, ADDEMUP, it was posited that successful components would maximize some linear combination of accuracy and diversity, with positive coefficients [Opitz and Shavlik, 1995] (described in Section 2.3.5). In ADDEMUP, the combiner computed a weighted sum of the component network outputs.

In this section we characterize empirically the relationship between these two criteria when applied to minimal nearest neighbor classifiers combined using ID3. Our experiment thus uses different component model classes and a different combining function from ADDEMUP. (The measure of diversity is also different.) The experimental evidence shows that the relationship between accuracy and diversity is complex and varies according to the data. Further, on some data sets, stacked generalizers with the highest displayed test accuracy did not incorporate components that maximized a linear combination of training accuracy and diversity.

Experiment. To study the relationship between accuracy and diversity for minimal nearest neighbor components, we performed the following experiment. For each of the 13 data sets, 100 stacked generalizers were generated by randomly selecting prototypes for each of two minimal nearest neighbor components. Each component classifier stored one prototype per class. Each stacked generalizer used ID3 as a combining algorithm. The data were split randomly into (90%) training and (10%) test sets.

The two independent variables in our experiment were the mean training accuracy of the components and their diversity on the training set. Diversity was defined as in Chapter 4, as the fraction of the training instances that are classified correctly by one component classifier, but not by the other. The dependent variable was the accuracy of the stacked generalizer on the test set. The experiment was asymmetric in that the independent variables were measured on the training set while the dependent variable was computed on the test set. This design was chosen to learn how stacked generalizers (actually) perform on the test data based on criteria that can be computed from the training set. Test set accuracy is the quantity of ultimate interest, but we must rely only on training data to try to maximize accuracy.

A three-tuple,

[component average accuracy, component diversity, composite accuracy],

was generated for each of the 100 stacked generalizers, and a contour graph of these 100 points was made for each data set. The contour graph shows the composite accuracy at various combinations of component accuracy and component diversity. For illustration, we show the graphs for the Soybean (Figure 5.1), Breast Cancer Wisconsin (Figure 5.2), LED-24 Digit (Figure 5.3) and Diabetes (Figure 5.4) data sets here; the graphs for all data sets are contained in Appendix B. In these graphs, the light areas depict regions of higher composite accuracy; the darker the region, the lower the composite accuracy. There are at least four observations that can be made.

(1.) Not all data sets have the same landscape. Qualitatively, some are characterized by smoothly graded landscapes, as in the Breast Cancer Wisconsin graph (Figure 5.2). Others, such as the Soybean graph (Figure 5.1), show interior peaks representing higher composite accuracy and some deep valleys of low accuracy.

(2.) In particular, the regions of the highest composite accuracy (the lightest regions) appear in different locations in the graphs. If it were true that the accuracy of these stacked generalizers was maximized through some linear combination of accuracy and diversity, then one would expect to find the lightest areas along some portion of the upper-right-hand edge of the depicted region (e.g., the graph for the Breast Cancer Wisconsin data set, Figure 5.2).

For some data sets, composite test accuracy is maximized at points in the interior of the graph (e.g., LED-24 Digit (Figure 5.3) and Soybean (Figure 5.1)). Interior points do not represent a maximal positive linear combination of accuracy and diversity: there are points in

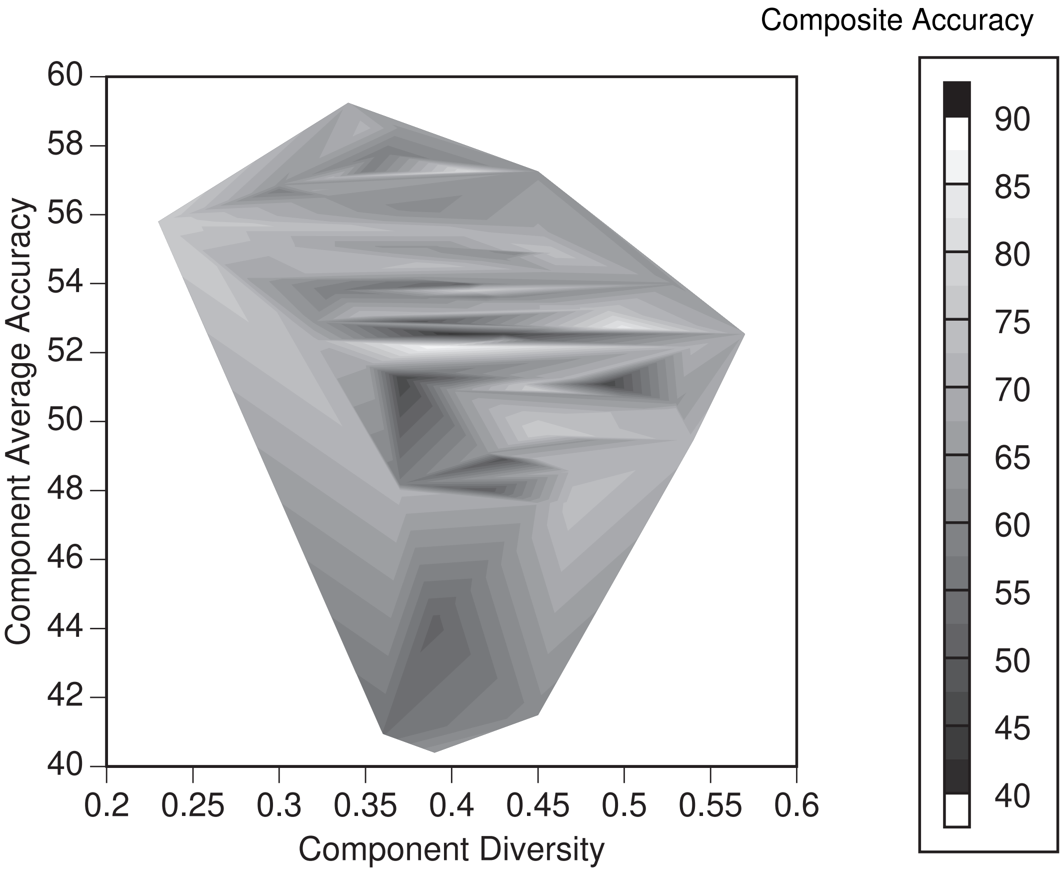

Figure 5.1: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Soybean data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class. Ignore the end intervals of the legend, which are supplied by the graphing program for contrast.

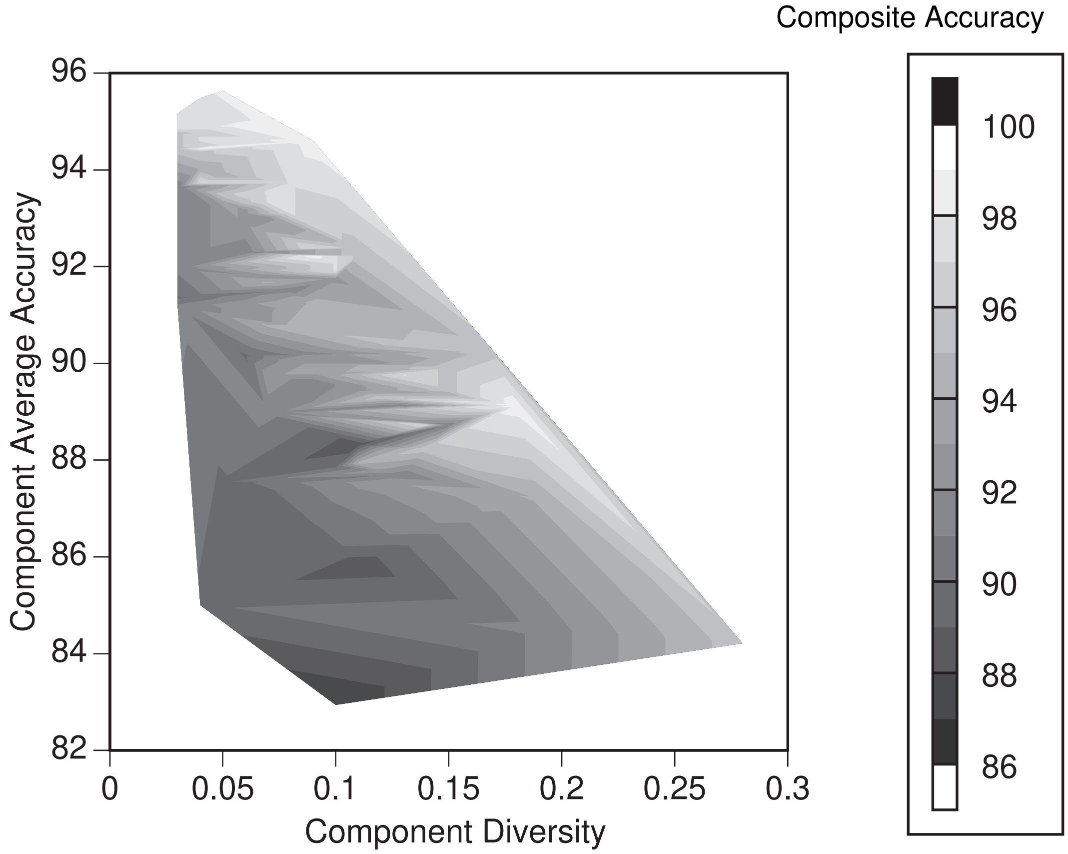

Figure 5.2: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Breast Cancer Wisconsin data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

the sample that are at least as high or higher on both dimensions. Yet, on a number of data sets, some interior points displayed the highest composite test accuracy.

For each data set, Table 5.9 [2] shows (a) the number of points of maximal composite test accuracy in the sample of 100 stacked generalizers; (b) the number of maximal points in (a) that appear in the interior of a graph; and (c) the number of the maximal points in (a) that appear on the (upper right-hand) boundary of a graph. The points in the interior of the graph correspond to stacked generalizers that did not display maximal component accuracy for a given level of diversity or that did not display maximal component diversity for a given level of accuracy (or did not display either of these characteristics). The points on the boundary correspond to stacked generalizers that did display maximal component accuracy and diversity in the sample.

Table 5.9: Location of points of maximal composite accuracy.

Data	*Composite Max. Acc.*	*Interior*	*Boundary*
Breast Cancer Ljubljana $_{280}$	3	2	1
Breast Cancer Wisconsin $_{690}$	7	4	3
Cleveland Heart Disease $_{300}$	2	0	2
Diabetes $_{760}$	1	1	0
Glass Recognition $_{210}$	1	0	1
Hepatitis $_{150}$	9	5	4
Iris Plants $_{150}$	1	1	0
LED-24 Digit $_{200}$	1	1	0
LED-7 Digit $_{500}$	1	0	1
Lymphography $_{140}$	10	8	2
Monks-2 $_{430}$	4	4	0
Promoter $_{100}$	1	1	0
Soybean $_{300}$			

Only three data sets exhibited the highest test accuracy at boundary points, where component diversity and accuracy were maximal (Cleveland Heart Disease, Glass Recognition

[2] On the training-set/test-set partition used for the Monks-2 data set, the stacked generalizer almost always attained equal accuracy, resulting in a large number of maximal points (97). Since almost all the points represent maximal accuracy, the number of interior or boundary points was not of particular interest and we did not compute their relative numbers.

and LED-7 Digit). For five data sets, maximal composite accuracy occurred only at interior points (Diabetes, Iris Plants, LED-24 Digit, Promoter and Soybean).

This experiment shows that for the model classes used and for some data sets, stacked generalizer test accuracy is not necessarily maximized by maximizing component accuracy and diversity. For other data sets, the only maximal points were seen at maximal levels of component accuracy and diversity, in fact.

In addition to these observations based on specific maximal *points*, composite accuracy also may be higher generally in *regions* where accuracy and diversity both tend to be lower. For example, for the LED-24 Digit data set, a region of relatively higher stacked generalizer accuracy can be seen at lower levels of both component accuracy and diversity (in the lower left corner of Figure 5.3).

(3.) Regions of high stacked generalizer accuracy appear quite close to regions of low accuracy. This characteristic would make the landscape a difficult one in which to do hill-climbing based on component accuracy and diversity; at the very least, the step size would have to be determined carefully. There are also peaks in some graphs that represent local maxima (e.g., the graph for the LED-24 Digit data, Figure 5.3).

(4.) The proximity of regions of high and low composite accuracy also reflects the partial inadequacy of component accuracy and diversity as dimensions for the space of stacked generalizers. It is possible that two component classifiers may have the same mean training accuracy and diversity and nevertheless yield stacked generalizers with different test set accuracies.

For example, on the Soybean data set, the components of two different stacked generalizers showed the same accuracy and diversity on the training set: 36.6% accuracy and 50.2% diversity. The stacked generalizers showed quite different accuracy on a test set: 41.9% and 61.3%. These composite accuracies are both very low, but the more important point is that training accuracy and diversity, at least as currently measured, do not completely determine test set accuracy.

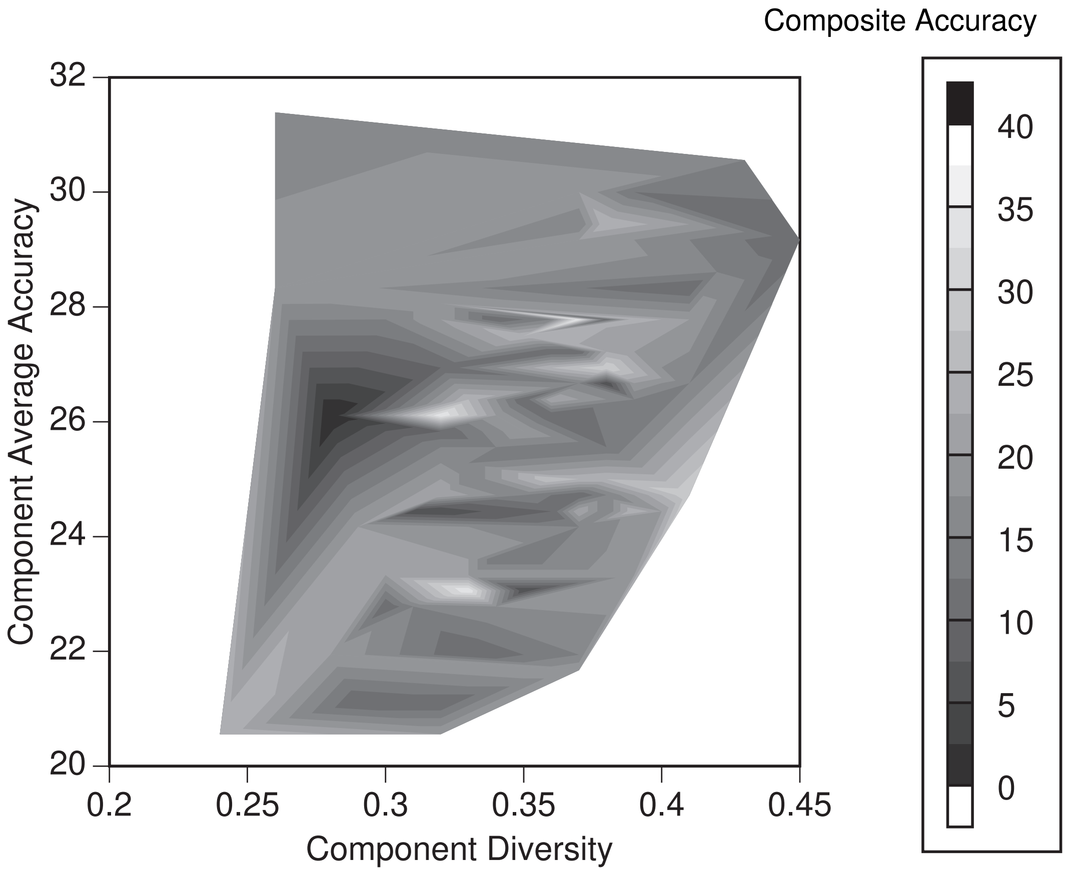

Figure 5.3: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the LED-24 Digit data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

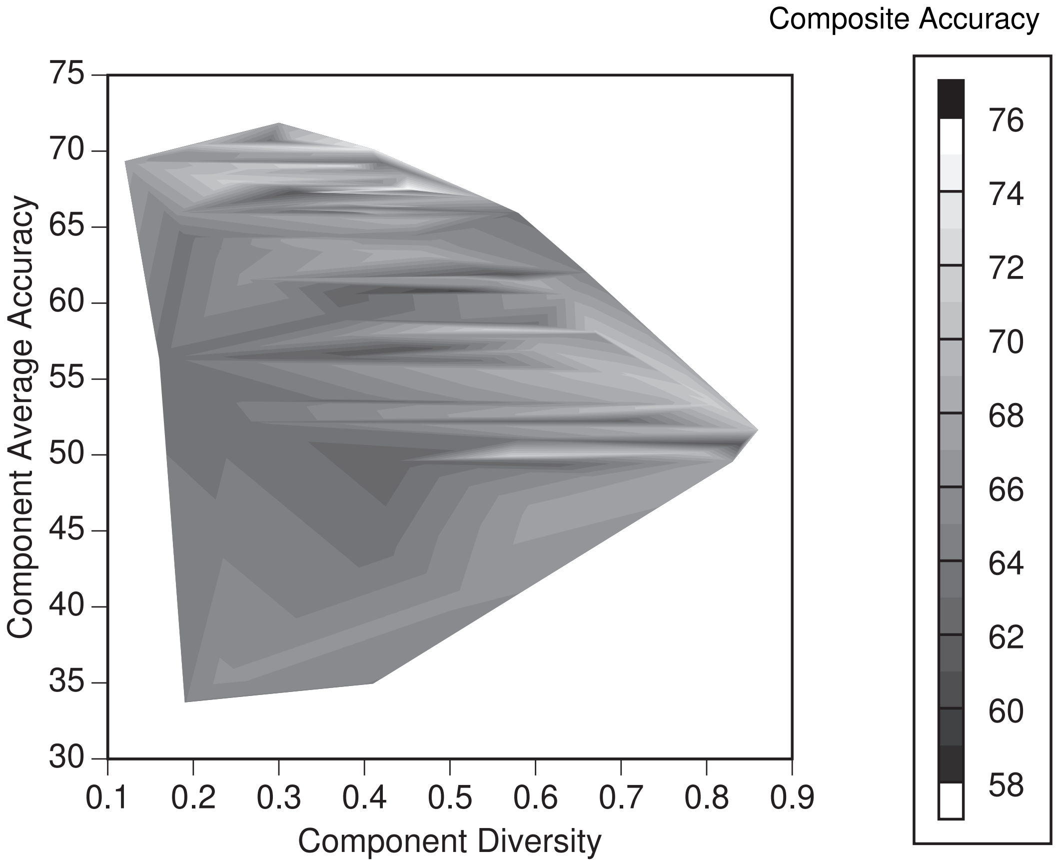

Figure 5.4: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Diabetes data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

At one level, this observation is not surprising. Any statistic computed on the training set may not reflect the behavior of a classifier on a test set that is independent. However, a standard assumption is that the training and test sets are drawn from the same distribution. In a theoretical computational learning setting, the test set might even have been created by an adversary, on the other hand. On another level, this observation shows that the dimensions of training accuracy and diversity, which appear to be useful criteria for composite accuracy, do not unambiguously characterize a stacked generalizer's accuracy on test data, even on commonly used data sets.

In summary, this experiment shows that the relationship between component accuracy and diversity, on one hand, and composite accuracy, on the other, makes a complex search space whose character depends on the data set. In particular, for some data sets, stacked generalizers of maximal test accuracy may not be found by maximizing component accuracy and diversity on the training set.

At this point in the chapter, we have provided experimental evidence that addresses some of the issues raised in the introduction to this chapter: the selection of a combining algorithm, the utility of using two minimal classifiers as components, the effect of combining a moderate number of component classifiers, and the relationship between accuracy and diversity for stacked generalizers with two components.

In the next section we show how a stacked generalizer can assess the probability that a class prediction is correct. We show that where the probability is low that a stacked generalizer's prediction is correct, a classifier can be designed that is less apt to be disastrously inaccurate. Where confidence in the composite prediction is low, the classifier outputs a component's prediction instead of the low-probability composite prediction.

5.4 Class Probability Estimates

In many pattern recognition and classification problems it is desirable to have a degree of confidence associated with a predicted class, a *class probability estimate* [Ripley, 1996]. Handwritten character recognition is a classical application in which a "reject" option is generally assumed to be available for low-confidence predictions [Xu *et al.*, 1992; LeCun *et al.*, 1989]. As a practical matter, when a test instance is rejected, it is given to a human for processing.

We have not insisted on class probability estimates in this research, but, because of their importance to some applications, we suggest one way to assess the level of confidence in a composite prediction. Our general approach is to take an undesirable side-effect of our architecture design — the class ambiguity of many level-1 training instances — and exploit it to create a confidence estimate for composite predictions.

We then can use the confidence estimate to create a composite classifier that is less apt to underperform the most accurate component. Where the confidence in a composite prediction is low, the composite classifier instead may output the prediction of the component that is the most accurate on the training set. Since this approach depends on having an accurate component classifier, it may be applied to stacked generalizers that incorporate a full nearest neighbor classifier component, for example.

In this section, we describe one straightforward way to compute a probability estimate for a composite prediction. We then give an algorithm that uses the probability estimate to improve accuracy when the composite prediction has low probability.

We have called a vector of predictions of the component classifiers for a level-0 instance its *level-1 feature representation*. For example, let there be two component classifiers, and let x be a level-0 instance. If the first component predicts class A when applied to instance x and if the second component predicts class B when applied to x, then the level-1 feature representation for x is (A, B). The complete level-1 representation for x, used to train the level-1 algorithm, also includes the class of x. If the class of x is A, the complete level-1 representation of x is (A, B, A).

As we discussed in Section 5.1, if level-0 instances from different classes have the same level-1 feature representation, then that instance representation is *ambiguous* (or *contradictory*). An ambiguous level-1 instance has the same feature values (the predictions of the component classifiers) but has different class labels. Ambiguous instances typically depress the accuracy of a learning algorithm, since most algorithms are unable to deal effectively with contradictory training data.

Ambiguous level-1 instances are one unfortunate consequence of our design for composite classifiers. Where only a small number of component classifiers are combined and where the level-1 feature representation consists only of vectors of class labels, there are likely to be many ambiguous level-1 instances. For example, where there are two classes, class A and class B, and two component classifiers, there are only four possible feature vectors available for a level-1 feature representation, $\{(A, A), (A, B), (B, A), (B, B)\}$. In effect, the component classifiers compress the data severely from their original level-0 representation into a level-1 feature representation. Due in part to the trauma of compressing the data so drastically, many instances with the same feature values and different class labels typically are present at the level-1 level, at least in the class of stacked generalizers that we investigate. Where the original level-0 instances are not available to the level-1 algorithm, there is information loss. (However, in a majority of Schaffer's experiments with *bi-level* stacking, in which the level-0 instances were also input to the level-1 algorithm, *lower* accuracy was obtained than with standard stacking [Schaffer, 1994]. See the discussion in Section 2.2.1.1.)

If a level-1 instance representation is very ambiguous — many level-0 instances from more than one class are mapped by the component classifiers to the same level-1 feature vector — then the confidence will be low for a prediction made for a level-0 instance with that level-1 feature representation. If a particular level-1 instance representation is very ambiguous, it will be difficult for the level-1 learning algorithm to make an accurate prediction. In this situation, it may be advisable to ignore the low-confidence prediction of the composite classifier and to substitute the prediction of the component with the highest training accuracy.

This general idea is to "retro-fit" an existing stacked generalizer by substituting the prediction of the most accurate component for the stacked generalizer's prediction, in certain circumstances. This modification is designed to make a stacked generalizer "safer" in the sense that the composite classifier is less likely to be substantially less accurate than the component that is the most accurate on the training set. Since the component's prediction is used where the composite prediction is evidently unreliable, at the very least, this procedure should yield predictions that are almost as accurate as the most accurate component.

We evaluate this idea experimentally through an evaluation of an algorithm based on this idea, which we call the *Safe* algorithm. Below, we give an example of how the Safe algorithm works and then formalize it. We apply the general Safe retro-fit to the Coarse Reclassification algorithm (Chapter 4) to create a "safe" variation of Coarse Reclassification, called *Safe Coarse Reclassification (Safe-CR).* Safe-CR less frequently displays accuracy lower than the base classifier it is trying to boost.

Example. Suppose there are two classes, A and B, and two component classifiers C_0 and C_1. Suppose ten instances from the training set have the level-1 feature representation (A, B), meaning that component C_0 predicts A and component C_1 predicts B when applied to each of the ten instances. Suppose further, that of these ten instances, four are from class A and six are from class B. Let us assume, as is likely, that the level-1 learning algorithm learns to predict class B for instances with level-1 feature representation (A, B). If the distribution of instances is the same on the training set and the test set, the composite classifier on average will correctly predict the class of instances with the level-1 feature representation (A, B) with 60% accuracy.

Suppose, on the other hand, that the predictive accuracy of component C_0 is 80% on the training set. For any instance whose level-1 feature representation is (A, B), the Safe algorithm ignores the composite prediction and outputs the prediction of C_0. For this algorithm to improve accuracy in this situation, the accuracy of the component has to be greater than 60% on instances that have the level-1 feature representation (A, B). For example, if the instances with representation (A, B) are in effect just a random sample of instances, then it is likely that 80% of them will be correctly classified by C_0. (Another, possibly better, "safe" algorithm would output the prediction of the component classifier that is most accurate for instances that have the level-1 feature representation (A, B), rather than the most accurate component for the entire training set. We have not explored this variation.)

When the prediction of the most accurate component is output by the stacked generalizer, stacked generalizer accuracy will track that component's accuracy. The stacked generalizer will do no worse than that component on such instances.

Algorithm. In a two-component classifier architecture, let $(S_{i_1}, S_{i_2}) \in S \times S$ be a level-1 feature representation, where S is the set of s class labels $S_i, i = 1, \ldots, s$. Let the two component classifiers be C_0 and C_1. Let n_{j,i_1,i_2} be the number of instances in the training set whose class is S_j and that have level-1 feature representation (S_{i_1}, S_{i_2}). Then the class probability estimate for the class S_i for a new instance with level-1 representation (S_{i_1}, S_{i_2}) may be defined as

$$p(S_i|(S_{i_1}, S_{i_2})) = \frac{n_{i,i_1,i_2}}{\sum_{j=1}^{s} n_{j,i_1,i_2}}$$

In the above example, the probability of class B, given a level-1 feature representation (A, B), is

$$p(B|(A,B)) = \frac{6}{4+6} = 0.6$$

Now, suppose a test level-0 instance has the level-1 feature representation (S_{i_1}, S_{i_2}). If

$$max\{p(S_i)|i = 1, \ldots, s\} < max\{\texttt{trainAccuracy}(C_0), \texttt{trainAccuracy}(C_1)\}$$

then the Safe algorithm will output the prediction of the more accurate component (instead of the prediction of the composite classifier). Otherwise, the prediction of the composite classifier will be output as usual.

This algorithm will not be useful where all of the components are known to have low accuracy, and might be better applied where one classifier is known to be quite accurate, as (is sometimes true) in boosting. In the following experiment, we evaluate the safe version of Coarse Reclassification, Safe-CR.

Experiment. The Coarse Reclassification algorithm was applied to create and train stacked generalizers with one full and one minimal nearest neighbor component, using ID3 as the combining algorithm. The prediction of the resulting stacked generalizer was modified according to the Safe algorithm. The mean test accuracy of Safe-CR is given in Table 5.10.

Analysis. Safe-CR shows significantly higher accuracy than Coarse Reclassification on two data sets: LED-24 Digit and Soybean. Importantly, these data sets are two on which Coarse Reclassification yields a stacked generalizer that is less accurate than the base nearest neighbor classifier. The Safe algorithm has substantially reduced the loss in accuracy on these data sets from ineffective combination. On the other hand, the Safe-CR algorithm has significantly lower accuracy than Coarse Reclassification on two data sets (Breast Cancer Wisconsin and Diabetes).

In summary, the Safe-CR algorithm shows significantly higher accuracy than Coarse Reclassification on three data sets where Coarse Reclassification performed poorly with respect to the base nearest neighbor classifier. In investment terms, Safe-CR offers a different risk/return profile. It is more conservative — less likely to make significant gains in accuracy, but more likely not to be less accurate than the most accurate component.

Table 5.10: Safe-CR algorithm mean test accuracy. Significance levels are given for the Safe-CR algorithm for improvement over the nearest neighbor baseline *(Sig.NN)*. Significance levels are also give for a difference in mean accuracy between Safe-CR and Coarse Reclassification *(Sig.CR)*.

Data	*NN*	*CR*	*Safe-CR*	*Sig. NN*	*Sig. CR*
Breast Cancer Ljubljana $_{280}$	67.5 ± 9.6	75.4 ± 6.6	74.3 ± 8.6	0.012	0.591
Breast Cancer Wisconsin $_{690}$	96.2 ± 1.8	97.8 ± 2.0	96.2 ± 1.8	0.500	0.032
Cleveland Heart Disease $_{300}$	77.0 ± 5.5	78.0 ± 9.5	78.0 ± 5.3	0.041	1.000
Diabetes $_{760}$	68.7 ± 5.1	73.7 ± 4.6	68.6 ± 5.3	0.661	0.012
Glass Recognition $_{210}$	73.3 ± 10.3	71.9 ± 10.6	73.8 ± 10.1	0.363	0.168
Hepatitis $_{150}$	77.3 ± 11.4	83.3 ± 7.2	83.3 ± 10.5	0.027	0.999
Iris Plants $_{150}$	91.3 ± 7.1	93.3 ± 4.4	91.3 ± 7.0	0.778	0.279
LED-24 Digit $_{200}$	36.5 ± 9.4	28.0 ± 7.2	32.5 ± 9.5	0.939	0.019
LED-7 Digit $_{500}$	71.2 ± 7.7	73.2 ± 8.6	73.2 ± 8.9	0.042	1.000
Lymphography $_{140}$	80.7 ± 8.9	75.0 ± 12.3	80.7 ± 10.1	0.500	0.104
Monks-2 $_{430}$	48.8 ± 4.7	66.7 ± 8.0	66.7 ± 7.9	0.000	1.000
Promoter $_{100}$	79.0 ± 9.9	80.0 ± 10.5	80.0 ± 10.5	0.171	1.000
Soybean $_{300}$	91.0 ± 3.9	86.0 ± 6.6	88.7 ± 6.9	0.967	0.010

In the final section of this chapter, we demonstrate a connection between stacked generalization and the k-nearest neighbor algorithm. We observe that a k-nearest neighbor classifier is a particular instantiation of a stacked generalizer that incorporates k component classifiers, each of which predicts the class of a different neighbor of a test instance. This observation provides a more general framework to investigate extensions to the k-nearest neighbor algorithm.

5.5 Generalizing the k-Nearest Neighbor Classifier

In Chapter 2, we observed that a major technical problem in this thesis, selecting prototypes for nearest neighbor classifiers that are to be combined, is at the confluence of two streams of research: nearest neighbor classifier design and classifier combination. In this section, we show that these two streams flow from the same source. From one perspective, a k-nearest neighbor classifier is a stacked generalizer. (A decision-tree, for example, is not a stacked generalizer, since not every node in the tree is applied to every level-0 instance, as in stacked generalization. A feedforward neural network trained with backpropagation is not a stacked generalizer, because the input units (analogous to the level-0 component classifiers) do not take an entire training vector as input, but only receive as input a single component of that vector. In addition, the input units do not output a prediction for the class of the (entire) input vector, as do the level-0 components in a stacked generalizer. Only the output units in the network yield a prediction. We have discussed other aspects of the relationship of stacked generalization to adaptive neural networks in Section 2.2.)

From one inconsequential perspective, a k-nearest neighbor classifier is a trivial stacked generalizer. A trivial stacked classifier consists of one k-nearest neighbor component classifier and the identity function (on class labels) as the combining algorithm.

A deeper connection can be made in another way. First, we distinguish a classifier that we have not found characterized in the research literature, which we (therefore) take the liberty of calling a *jth-nearest neighbor* classifier. Second, we observe that these classifiers can be combined to create a k-nearest neighbor classifier.

A jth-nearest neighbor classifier is an instance-based classifier that predicts the class of the jth-closest prototype. Procedurally, the j-nearest neighbors are computed as for a j-nearest neighbor classifier, but rather than amalgamating the predictions of the neighbors by majority vote, a jth-nearest neighbor classifier simply predicts the class of the jth-closest stored training instance. For example, a 2nd-nearest neighbor classifier predicts the class of the instance that is the second closest to a test instance.

This idea is analogous to auto-correlation for time series analysis [Chatfield, 1984]. Time series observations at a lag greater than one (observations before the immediately preceding observation) may have a greater statistical correlation with the current observation than the immediately preceding one. Periodic series often have this property. By analogy, the class of instances that are farther from the test case may have a higher correlation with the class of test case than the class of the nearest neighbor.

To test the accuracy of jth-nearest neighbor algorithms, we computed the test accuracy of jth-nearest neighbor classifiers that store all training instances as prototypes, for $j = 1, \ldots, 5$. As usual, ten-fold cross validation was used and the mean test accuracy is given. Table 5.11 shows the mean accuracies and the data sets for which 1st-nearest neighbor was superior at the $p \leq 0.05$ significance level. In only approximately one-third of the experiments was the 1-nearest neighbor statistically superior to the jth-nearest neighbor for $j = 2, \ldots, 5$. Thus the jth-nearest neighbor is often as accurate as the 1-nearest neighbor algorithm.

For one data set, Monks-2, the jth-nearest neighbor algorithm was significantly more accurate than the 1-nearest neighbor algorithm. In the Monks-2 problem, the target concept has exactly two 1's in a feature vector of length six. Neighbors that are at a minimum distance (1) from a test instance are often of the opposite class. Thus, the closest neighbors to concept members are frequently incorrect, and the nearest neighbor can be a poor predictor. (The order of the instances in the data also affects nearest neighbor accuracy on the Monks-2 data.) Other concepts also show regularity in the correlation between the classes of instances at different distances. Parity is one obvious example.

Table 5.11: Test accuracy for the *jth*-nearest neighbor algorithm, $j = 1, \ldots, 5$. "*" indicates that the 1-nearest neighbor algorithm is significantly ($p \leq 0.05$) more accurate than the *jth*-nearest neighbor classifier; "†" indicates that the *jth*-nearest neighbor classifier is significantly more accurate ($j = 2, \ldots, 5$) than the 1st-nearest neighbor.

Data Set	*1st-nn*	*2nd-nn*	*3rd-nn*	*4th-nn*	*5th-nn*
Breast Cancer Ljubljana $_{280}$	67.5	66.8	65.4	65.7	65.0
Breast Cancer Wisconsin $_{690}$	96.2	95.7	96.2	95.2	94.1*
Cleveland Heart Disease $_{300}$	77.0	78.7	70.7*	76.3	73.7
Diabetes $_{760}$	68.7	69.6	64.9*	68.0	67.8
Glass Recognition $_{210}$	73.3	74.3	65.7*	59.0*	61.0*
Hepatitis $_{150}$	77.3	76.7	77.3	84.0	74.0
Iris Plants $_{150}$	91.3	96.7	92.7	93.3	92.7
LED-24 Digit $_{200}$	36.5	38.5	37.5	28.0*	27.5*
LED-7 Digit $_{500}$	71.2	63.8*	63.8*	68.0	59.8*
Lymphography $_{140}$	80.7	71.4*	78.6	74.3	70.7*
Monks-2 $_{430}$	48.8	68.1†	57.0†	56.0†	53.3
Promoter $_{100}$	79.0	75.0	73.0	69.0	71.0*
Soybean $_{300}$	91.0	85.3*	75.0*	75.7*	71.0*

The second step establishes the connection: combining jth-nearest neighbor classifiers by voting is tantamount to a k-nearest neighbor classifier. Suppose that we have k component jth-nearest neighbor classifiers, where $j = 1, \ldots, k$. Supppose, further, that all of the jth-nearest neighbor classifiers store the same set of prototypes. Then, if plurality vote is used to combine the predictions of these k component classifiers, the classical k-nearest neighbor algorithm is obtained (Figure 5.5).

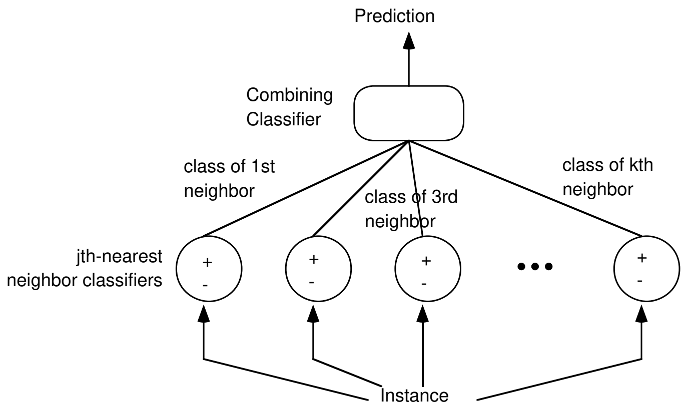

Figure 5.5: Stacked generalizer with k component *jth*-nearest neighbor classifiers, $j = 1, \ldots, k$, as a generalization of a k-nearest neighbor classifier.

This interpretation suggests at least two ways to generalize the classical k-nearest neighbor algorithm: (1) apply different combining functions to combine the predictions of the k closest neighbors, and (2) store different sets of prototypes in the jth-nearest neighbor classifiers. This interpretation provides an analogy to the more general framework provided by case-based reasoning (CBR). In CBR, sophisticated techniques are typically used for analyzing and reasoning with the similar cases, which may vary from case to case and application to application [Rissland *et al.*, 1984; Rissland, 1989]. In addition, case-based reasoners will perform differently depending on the cases that are stored in their case bases and depending on the importance that is (statically or dynamically) attached to individual cases or case prototypes.

In this next experiment, we investigate generalization (1): the effect of using an alternative combining algorithm for the predictions of the jth-nearest neighbor component classifiers. This approach for symbolic predictions is analogous to a well-studied situation where the components are function approximators that predict real values [Atkeson *et al.*, 1996]. In that setting, a regression function may be used to fit the real-valued predictions of the nearest neighbors. For example, a weighted average of the neighboring predictions may be used to give a prediction for a test instance.

Experiment. For each data set, we constructed a stacked generalizer consisting of three jth-nearest neighbor component classifiers, for $j = 1, 2, 3$, and ID3 as the level-1 combination algorithm. Each component stored all training instances as prototypes. Here, the appropriate baseline algorithm is a 3-nearest neighbor algorithm, since the stacked generalizer also has access to three neighbors.

Analysis. This approach failed to show a significant improvement at the $p \leq 0.05$ level for most of the data sets as shown in Table 5.12, and shows poor results on at least three data sets (Iris Plants, LED-24 Digit and Lymphography).

However, significant improvement was seen on three data sets (Breast Cancer Wisconsin, Monks-2 and Promoter). Notably, the results for these three data sets are the highest attained by any of the stacked generalizers considered in this thesis. For those three data sets, there is apparently some regularity in how the jth-nearest neighbor's class correlates with the true class

Table 5.12: Comparison of a 3-nearest neighbor algorithm with a stacked generalizer incorporating three *jth*-nearest neighbor components, $j = 1, 2, 3$.

Data	*3-nn*	*jth-NN*	*Significance*
Breast Cancer Ljubljana $_{280}$	71.4 ± 10.5	72.1 ± 6.7	0.392
Breast Cancer Wisconsin $_{690}$	96.7 ± 1.4	97.4 ± 2.0	0.048
Cleveland Heart Disease $_{300}$	79.3 ± 5.4	78.7 ± 6.5	0.828
Diabetes $_{760}$	70.7 ± 5.3	70.7 ± 4.9	0.500
Glass Recognition $_{210}$	71.9 ± 11.1	71.4 ± 9.3	0.560
Hepatitis $_{150}$	80.0 ± 8.3	78.7 ± 7.6	0.669
Iris Plants $_{150}$	95.3 ± 5.5	93.3 ± 7.0	0.959
LED-24 Digit $_{200}$	45.0 ± 13.3	21.0 ± 6.2	1.000
LED-7 Digit $_{500}$	72.0 ± 8.7	69.8 ± 7.1	0.907
Lymphography $_{140}$	82.9 ± 10.2	80.0 ± 8.1	0.982
Monks-2 $_{430}$	48.6 ± 3.5	86.3 ± 6.0	0.000
Promoter $_{100}$	77.0 ± 12.5	84.0 ± 9.7	0.033
Soybean $_{300}$	86.7 ± 6.1	87.0 ± 5.5	0.438

of an instance, which is not exploited by using majority vote of the neighbors to make a class prediction, according to the usual k-nearest neighbor algorithm. This regularity is learned by the ID3 combining algorithm.

In this section we have shown that a stacked generalizer with nearest neighbor components is a generalization of a k-nearest neighbor classifier. For three of the data sets in this test suite, this observation led to stacked generalizers that have higher accuracy than has been obtained by other algorithms in this thesis.

5.6 Summary

In this chapter we have tried to provide answers to some obvious questions about stacking minimal nearest neighbor component classifiers.

We began with a set of experiments to determine the accuracy of different combining algorithms. We showed that, in general, voting is less accurate than ID3 and than a nearest neighbor algorithm where the components are random minimal nearest neighbor classifiers. Where the components are the most accurate minimal nearest neighbor classifiers in a given sample, voting was generally comparable to ID3 and to a nearest neighbor algorithm as a level-1 combining function.

We then demonstrated that, on approximately half of the data sets in our collection, combining the predictions of a set of minimal nearest neighbor components was significantly more accurate than a full nearest neighbor classifier. Even where only two minimal nearest neighbor components were combined, using ID3, significant improvement was also seen on five data sets. As more components were added, stacked generalizer accuracies improved monotonically on three data sets, but on two other data sets, accuracies were seen to decrease with an increase in the number of components.

We continued our exploration of the dimensions of component accuracy and diversity, and showed graphically that the relationship is quite complex, even for two minimal nearest neighbor component classifiers. It was demonstrated empirically that the stacked generalizers that were the most accurate on test data did not necessarily maximize a (non-negatively-weighted) sum of training component accuracy and diversity. Stacked generalizers with the highest observed test accuracy frequently possessed lower component accuracy and lower diversity than less accurate stacked generalizers.

We raised the question of whether class probability estimates could be made for stacked generalizer predictions. The question was raised partly through an attempt to determine whether a more accurate prediction might be made when it was clear that the stacked generalizer would only be able to provide a low-confidence prediction. We demonstrated that, where the level-1 representation of an instance is highly ambiguous, outputting the prediction of the most accurate component can be more accurate than the stacked generalizer's prediction.

Finally, we have shown how the classical k-nearest neighbor classifier is a type of stacked generalizer. This demonstration suggested that a different combining function could be used to combine the symbolic predictions of the k nearest neighbors. Combining the predictions of the three closest neighbors using ID3 showed the highest accuracies achieved in this dissertation on three data sets.

CHAPTER 6

CONCLUSIONS

This chapter summarizes the dissertation, discusses its contributions and suggests directions for future research.

6.1 Summary of the Dissertation

Our goal in this thesis has been to evaluate the hypothesis that *the accuracy of a standard nearest neighbor classifier can be improved or exceeded through composite-classifier architectures that incorporate a small number of diverse component nearest neighbor classifiers, each of which stores only a small number of prototypes.* For 10 of the 13 representative data sets we have examined, we have introduced at least one algorithm that significantly outperforms a nearest neighbor algorithm. These algorithms provide a constructive, empirical demonstration that the hypothesis is true with respect to 10 of these data sets. On three data sets (Glass Recognition, Lymphography and Soybean), none of the algorithms studied significantly improved nearest neighbor accuracy. With respect to these three data sets, therefore, there is evidence that the hypothesis is not true.

There are three areas in which we have shown progress: prototype selection, boosting and stacked generalization. Here we summarize the primary conclusions. Our investigations have been quite wide-ranging, and a variety of secondary results were established in the chapters dedicated to the three areas (Chapters 3, 4, and 5).

Prototype Sampling. Prototype sampling was introduced in Chapter 3 as a response to the failure of previous editing algorithms to search extensively the space of small sets of prototypes. In this chapter, prototype sampling was applied to the problem of selecting a minimal set of prototypes for a single, independent nearest neighbor classifier. On a majority of the data sets in our collection, it was shown that drawing a rather small number of samples of prototype sets (100) yielded nearest neighbor classifiers that were more accurate or not significantly less accurate than a nearest neighbor classifier that used all available training instances as prototypes. Prototype sampling reduces by as much as two orders of magnitude the number of inter-instance distance computations needed to classify new instances.

Sampling nearest neighbor classifiers was applied to generate a new taxonomy of instance types. The relationship between two quantities was studied: (a) the accuracy of nearest neighbor classifiers when they were applied to each instance and (b) their average accuracy on the entire training set. Exploring this relationship led to the conclusion that each of the data sets contained instances that were positively correlated with *lower* classification accuracy in the following sense. If that instance was *correctly* classified by a minimal nearest neighbor classifier, the overall accuracy on the training set was *lower*, on average, than if the instance were *incorrectly* classified. These *uncharacteristic* instances were removed from the validation set that was used to test the accuracy of sampled nearest neighbor classifiers. No statistically significant loss in accuracy was observed, and testing on fewer instances was more efficient, once the uncharacteristic instances were identified.

Boosting. We turned to the central question of the dissertation: whether it is possible to increase nearest neighbor accuracy through prototype selection for a set of nearest neighbor classifiers whose predictions are to be combined. Boosting was the first composite architecture studied, where it was assumed that a nearest neighbor classifier was given as a level-0 component (the *base classifier*). The base classifier was assumed to be a 1-nearest neighbor classifier that stored all training instances as prototypes, and that applied a Manhattan distance metric to real-valued features and an exact-match distance metric to symbolic features.

The task was to create a complementary, minimal nearest neighbor classifier and to combine its predictions with those of the given classifier. The objective was for the composite classifier to increase or *boost* the accuracy of the base nearest neighbor classifier. Achieving this objective required that the complementary classifier make predictions that were different from the base nearest neighbor classifier. The diversity-inducing technique also had to allow the level-1 learning algorithm to learn the correct prediction from the two component predictions.

In our study of boosting, we presented four algorithms, which were called *Coarse Reclassification*, *Deliberate Misclassification*, *Composite Fitness* and *Composite Fitness–Feature Selection*. Four algorithms were investigated to explore the effectiveness of a variety of techniques for creating diverse classifiers, as summarized in Table 6.1.

Table 6.1: Techniques used by four nearest neighbor boosting algorithms to create diverse component classifiers.

Algorithm	*Technique*
Coarse Reclassification	sampling minimal classifiers
Deliberate Misclassification	modification of training instance class targets
Composite Fitness	maximization of composite training accuracy
Composite Fitness–Feature Selection	feature selection

This investigation of four different techniques for creating diversity for components from a single model class is the most comprehensive investigation that we have encountered. Three primary conclusions were drawn. First, the four techniques generally were successful at boosting nearest neighbor accuracy on a core of four data sets. In terms of demonstrating statistically significant improvement over a nearest neighbor classifier, the best performance was by the Deliberate Misclassification algorithm, which showed significant improvement on five data sets in the collection. Deliberate Misclassification applied a new technique for inducing diversity: selectively changing the class labels of certain training instances.

Second, significant improvements were not achieved by any of the four algorithms on four data sets (Glass Recognition, Lymphography, LED-24 Digit and Soybean). Although not conclusive evidence, these experiments suggest that a single minimal nearest neighbor component classifier is not sufficiently powerful to boost nearest neighbor accuracy on these four data sets. An investigation of the characteristics of these four data sets showed that they exhibit the smallest average number of instances per class in this collection. We speculated that a lower density of instances per class made prototype sampling less effective. A lower density of instances affects the proximity of instances in the instance space.

Third, the four boosting algorithms were seen to be selectively superior. Each of the four boosting algorithms outperformed the other three on at least one data set. An examination of the degrees of component accuracy and diversity attained by the four algorithms reveals that Coarse Reclassification and Deliberate Misclassification typically displayed similar measures of accuracy and diversity. Different measures of component diversity and accuracy were typically attained by the Coarse Reclassification, Composite Fitness and Composite Fitness-Feature Selection algorithms. Since the algorithms frequently demonstrated quite similar boosted accuracies, this evidence suggested that increasing composite accuracy is not a matter of finding a single combination of component accuracy and diversity that leads to increased accuracy.

Stacked Generalization. After investigating boosting architectures, we turned to stacked generalization. Our primary focus was on stacked generalizers that incorporated only minimal nearest neighbor classifiers as components. It was shown that voting was a poor combiner for small sets of *random* components. Voting was approximately as effective a combiner as a nearest neighbor algorithm and as ID3, where the components were the *most accurate* in a given sample.

For several data sets, such as LED-24 Digit, Promoter and Soybean, applying a decision-tree algorithm such as C4.5 directly to the original instances yielded more accurate results than using ID3 to combine a set of minimal classifiers. For other data sets (Breast Cancer Wisconsin, Cleveland Heart Disease and Diabetes), a stacked generalizer that combined only two minimal classifiers with ID3 outperformed C4.5.

Our design decision to limit the number of classifiers was temporarily relaxed to study the effect of combining a somewhat larger number of classifiers by voting. On approximately half of the data sets, composite classifiers that used voting to combine a larger number of classifiers outperformed both a full nearest neighbor classifier and the median component accuracy. We argued that median accuracy was a reasonable baseline in view of the application of voting as a combining mechanism.

One general drawback of the stacked generalization algorithms that we presented is that the composite classifier may be less accurate than the most accurate component. To create a stacked generalizer that is less likely to make errors that are not made by the most accurate component, we investigated an amendment to the method by which a fully-trained stacked generalizer makes its prediction. When a test instance's level-1 feature representation is highly ambiguous, the prediction of the most accurate component is output in place of the usual stacked generalizer prediction. A level-1 instance representation is ambiguous when the component classifiers generate the same vector of predictions for instances from different classes. On three data sets where a stacked generalizer generated by the Coarse Reclassification algorithm was less accurate than a nearest neighbor classifier, this amendment yielded statistically significant improvements in accuracy over Coarse Reclassification. This result showed one means to decrease the extent to which a composite classifier may be less accurate than the most accurate component.

We investigated empirically the relationship between (a) component training accuracy and diversity and (b) composite test accuracy. We evaluated the intuition that the stacked generalizers with the highest accuracy in a sample would reflect the highest component accuracy and diversity. This reasonable intuition proved to be incorrect. On ten of the 13 data sets, less accurate composite classifiers showed equal or greater degrees of both component accuracy and diversity than the most accurate composite classifiers. This result has implications for how the search is performed for an accurate stacked generalizer. For some level-0 model classes and level-1 combining algorithms, directing the search by attempting to maximize a combination of component accuracy and diversity may not lead to the most accurate stacked generalizers.

Finally, it was demonstrated that a k-nearest neighbor classifier is in fact a stacked generalizer that combines a set of so-called *jth-nearest neighbor* classifiers. This observation led to an alternative method for combining the predictions of the k nearest neighbors, since the rule for combining the predictions is in effect a level-1 combining algorithm. When ID3 was used to combine the predictions of the nearest neighbors, three data sets displayed the highest accuracies attained with minimal nearest neighbor component classifiers in this dissertation. This evidence suggests that on some data, majority voting does not adequately exploit the regularity in the classes of the k nearest neighbors.

We can also summarize some of this research in a second way. In our attempt to find nearest neighbor components that met our design criteria, we have searched portions of the space of k-nearest neighbor classifiers. This space may be may be considered a four-dimensional space, in which a k-nearest neighbor classifier may be characterized by a four-tuple: (P, k, d, θ), where

P is a set of prototypes,
k is the number of neighbors in a neighborhood,
d is the distance function, and
θ is the rule that combines the class predictions of a set of k nearest neighbors to yield a final prediction for a test instance.

Our primary focus has been on P, the problem of finding prototypes for a set of nearest neighbor classifiers, which is an extension of the classical problem of editing the set of instances in a single nearest neighbor classifier [1].

This research has also began to explore two of the other constituents of a nearest neighbor classifier: the distance metric (d) and the rule for combination of the predictions of the closest neighbors (θ). In the Composite Fitness–Feature Selection algorithm, the distance metric d was modified to compute the similarity of two instances only with respect to one feature. It was shown that the minimal nearest neighbor classifiers using that metric were highly inaccurate, but nevertheless significantly boosted accuracy of a nearest neighbor classifier on five data sets.

The rule (θ) for combining a set of neighbors was also explored briefly when ID3 was substituted for the majority rule. This substitution was made in light of our observation that a k-nearest neighbor classifier is a type of stacked generalizer that combines jth-nearest neighbor classifiers.

A large number of experiments were performed in connection with our investigation of the central idea of this dissertation — that combining minimal nearest neighbor classifiers is a useful approach to classification. Taken individually, each experiment demonstrated a specific point that contributed to our understanding of this idea and to the fundamental considerations

[1] Where both k and P should be specified explicitly, the term "(P, k)-nearest neighbor classifier" might usefully refer to a k-nearest neighbor classifier that incorporates P, a set of prototypes.

behind it. Taken together, these experiments allow us to provide some broad brush, heuristic guidance on the utility of the composite algorithms we investigated.

All of the boosting algorithms added one complementary minimal nearest neighbor classifier to a full nearest neighbor classifier. Within this framework, the Coarse Reclassification algorithm was the simplest. While the other three boosting algorithms enjoyed higher accuracy on some data, Coarse Reclassification performed as well as any of them across the board. Informally, this simple algorithm may reap most of any benefit to be gained from adding a single, minimal, complementary component classifier. Where it is important that the composite classifier not be any less accurate than a full nearest neighbor classifier, the Safe-Coarse Reclassification can be recommended as a way to limit the "downside" risk of reducing accuracy. In general, Coarse Reclassification performed better on two-class data, and comes less well recommended for multi-class problems. On two-class data, users may also wish to evaluate the utility of creating a stacked generalizer that combines a small number of minimal classifiers.

6.2 Contributions

Despite the research done on the combination of classifiers, little is definitely known about what classifiers may be combined effectively, and, if so, how. Generally, many learning algorithms have been developed, and each displays selective superiority over others on some data sets [Brodley, 1994]. Rather than focusing attention on developing still more learning algorithms, researchers have asked with increasing frequency whether there is merit in combining known classifiers to achieve higher accuracy. The hope is that perhaps combining even simple and efficient classifiers will lead to accuracy higher than some of the more sophisticated algorithms. Alternatively, one of the sophisticated algorithms might be combined with a simple one to increase inaccuracy. This thesis has been devoted to pursuing these speculations.

One of the oldest and best known classifiers is the nearest neighbor classifier. Despite its simplicity, the nearest neighbor classifier has proved to be effective in a variety of pattern recognition applications. Little research has investigated the hypothesis that combining nearest neighbor classifiers will increase accuracy over a single nearest neighbor classifier. Previous research has shown that for at least two composite architectures, combining nearest neighbor classifiers that incorporate a large number of prototypes does not lead to increased accuracy.

Our contribution is to have shown that, on a variety of commonly studied data sets, nearest neighbor classifiers that store only a small number of prototypes can be used effectively as components in composite classifiers. In Chapters 4 and 5, we have provided evidence that some of the hope for classifier combination might be realized. Very simple classifiers may be used, either alone or in conjunction with more sophisticated ones, to improve accuracy on a variety of data sets. We have gathered this evidence with respect to nearest neighbor classifiers. By deliberately introducing instability into the classifiers that are to be combined, we have overcome some of the problems associated with combining nearest neighbor classifiers, which tend to be stable when many prototypes are stored.

Most classifier combination research has been aimed at combining larger sets of models or at combining models that generate complex hypotheses, or both. This dissertation has explored a different region of the implicit search space of composite classifiers, the region in which a small number of simple classifiers are combined, rather than a large number of complex ones. In addition, our results help to answer the question raised by researchers in this field: whether a composite algorithm can be designed that leads to an efficient level-1 representation [Kong and Dietterich, 1995]. In short, this thesis suggests an alternate path towards accurate and efficient classification.

6.3 Issues for Future Research

In this section we suggest ways that this research may be extended.

Negative results for certain data sets. Our attempts at boosting four data sets met with failure. While we have shown that this failure is correlated with the number of instances in each class, future research might be directed to assigning blame for this failure. Some other algorithm might be designed to create a single, complementary minimal nearest neighbor classifier to serve as an adequate boosting component. On the other hand, it might be shown that minimal classifiers are inadequate for boosting a nearest neighbor classifier on these four data sets. The hypotheses they create may be inherently too coarse to improve a more complex classifier.

Classifiers from other model classes. To show that nearest neighbor classifiers may be combined effectively, we have limited our attention to that model class. A next step would be to try to apply some of the methods developed in this dissertation to other local classifiers that explicitly rely on neighborhood construction, such as radial basis function (RBF) classifiers. Prototype sampling algorithms may be useful for setting the centers for RBF networks. An additional step may be to determine whether the combination of coarse-hypothesis classifiers from other model classes may be combined effectively or used as complementary component classifiers for boosting.

Searching the space of nearest neighbor classifiers. As we observed at the end of the previous section, we have begun to investigate several dimensions of the search space of k-nearest neighbor classifiers in the attempt to produce diverse components. Much more work might be done to explore the rule for combining the nearest neighbors, however. Some learning problems, such as parity, display regularities in how an instance's class correlates with the classes of the neighbors. For example, in parity, the neighbors at (Euclidean) distance one are of the opposite class, while neighbors at distance two are of the same class. Varying the rule for combining the predictions, as is done for local classifiers with real-valued predictions [Atkeson *et al.*, 1996], may improve classification accuracy of an individual classifier. Additionally, combining classifiers that apply internally different rules for combining the predictions of a neighborhood of instances, may yield the diversity from which a level-1 algorithm may learn.

Incremental algorithms. All of the algorithms that we have presented are batch algorithms, where the entire training set is available in one pass, and the composite classifier is constructed once in batch from the training set. There are a number of advantages to incremental algorithms, though. Incremental algorithms, which incorporate each piece of data as it is presented to the agent, need not store all previously seen instances. As a result, incremental algorithms often can respond more readily to concept drift, where the underlying concept varies with time.

Taxonomy of instances. In Section 3.4.3 we proposed a taxonomy of instance types, based on statistics collected by applying a sample of classifiers to each instance. There are many ways in which that work might be investigated more fully. One direction, of potential interest to case-based reasoning researchers and cognitive scientists, would be to create a taxonomy of examples in a well-understood domain. Particular cases that fall into each taxonomic category could then be identified and explored. This study may help us to understand better the connection between different types of examples and the diversity required for effective classifier combination.

APPENDIX A

INSTANCES RANDOMLY OMITTED FROM CROSS-VALIDATION

Table A.1 specifies for each data set the indexes of the instances that were randomly omitted from the cross-validation partitions used in the experiments in this thesis. Instances were omitted to achieve cross-validation plies of equal cardinality, so as to enable the application of any statistical significance tests that may require equal-size plies. Each index specifies the line number of the omitted instance in the data file as it appears in the U.C.I. Machine Learning Repository, beginning with line 0 as the first line.

Table A.1: Instances randomly omitted from cross-validation.

Data	*Num. Omitted*	*Instances Omitted*
Breast Cancer Ljubljana $_{280}$	6	22,58,95,120,129,264
Breast Cancer Wisconsin $_{690}$	9	80,100,258,355,435,553,599,604,625
Cleveland Heart Disease $_{300}$	3	69,164,277
Diabetes $_{760}$	8	15,63,66,183,336,351,543,755
Glass Recognition $_{210}$	4	63,110,154,212
Hepatitis $_{150}$	5	7,40,46,115,144
Iris Plants $_{150}$	0	
LED-24 Digit $_{200}$	0	
LED-7 Digit $_{500}$	0	
Lymphography $_{140}$	8	35,47,71,82,91,106,142,145
Monks-2 $_{430}$	2	87,338
Promoter $_{100}$	6	7,22,25,44,69,73
Soybean $_{300}$	7	21,66,72,153,198,243,285

APPENDIX B

ACCURACY AND DIVERSITY CONTOUR GRAPHS

Below are the component accuracy and diversity contour graphs referenced in Section 5.3.

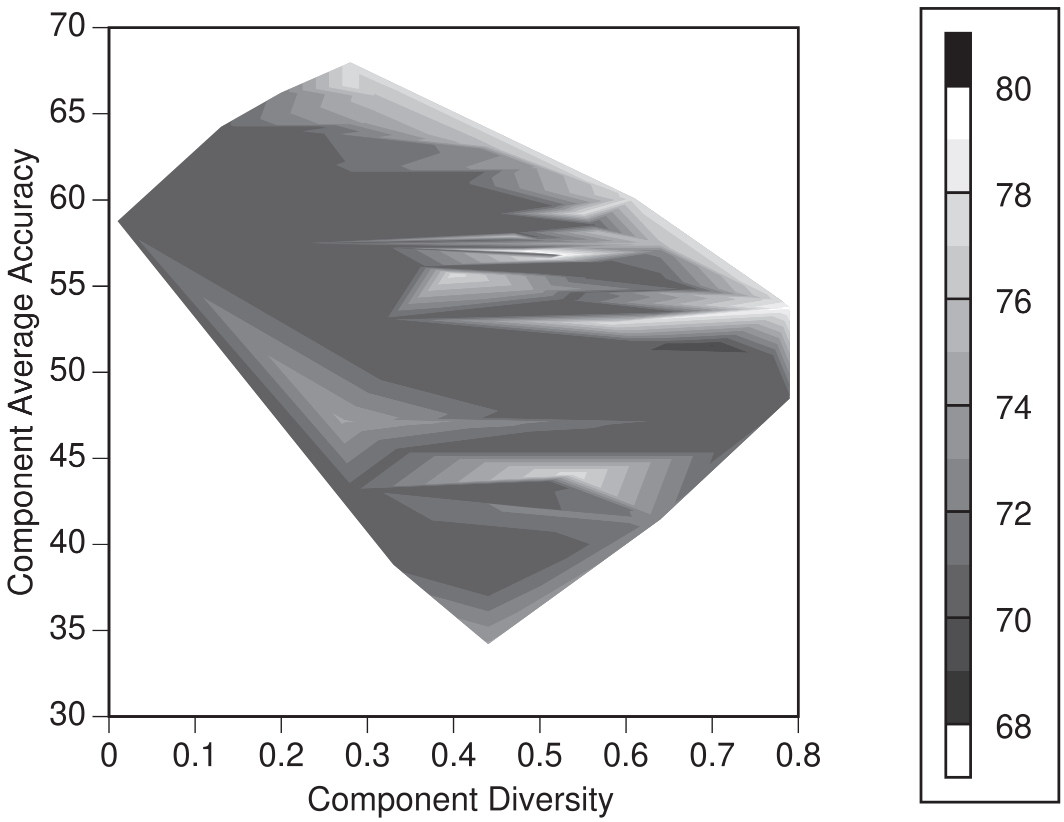

Figure B.1: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Breast Cancer Ljubljana data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

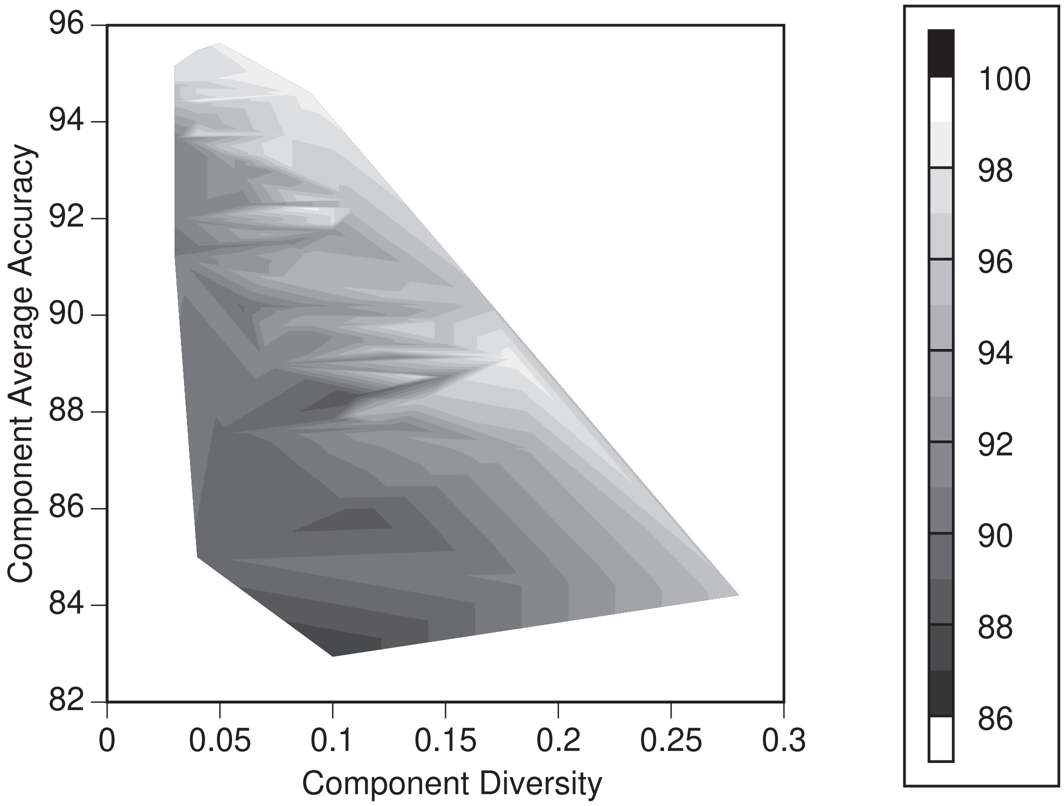

Figure B.2: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Breast Cancer Wisconsin data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

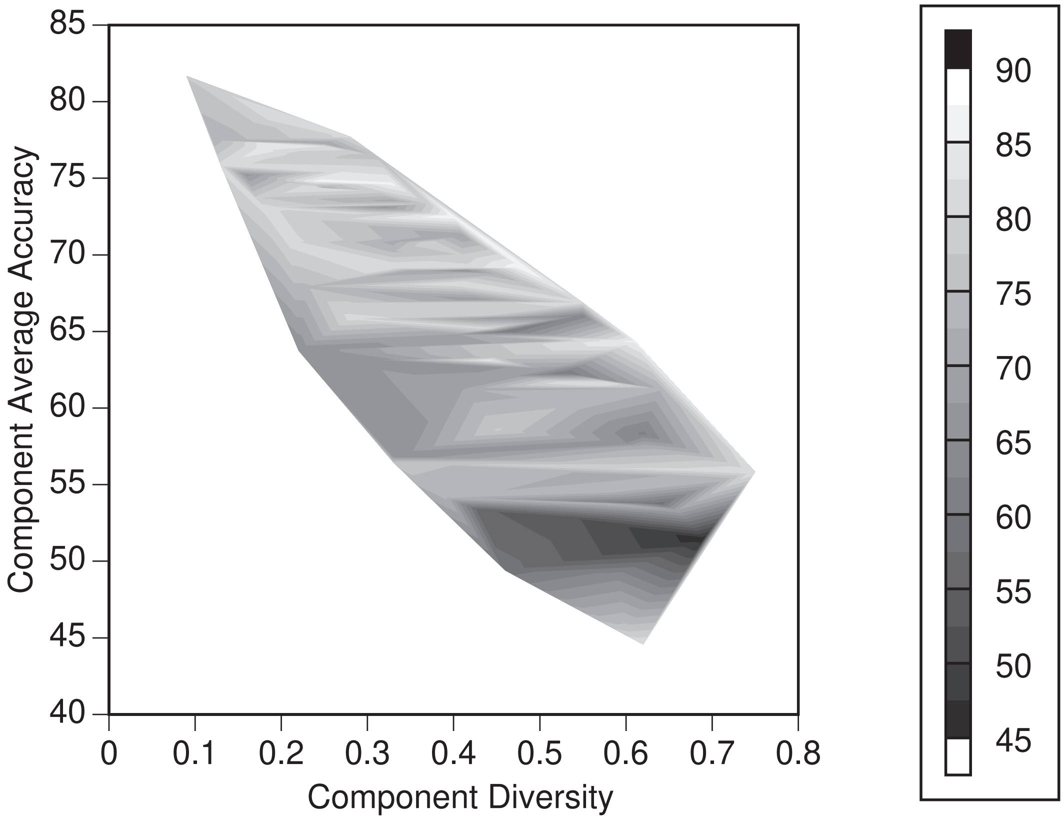

Figure B.3: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Cleveland Heart Disease data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

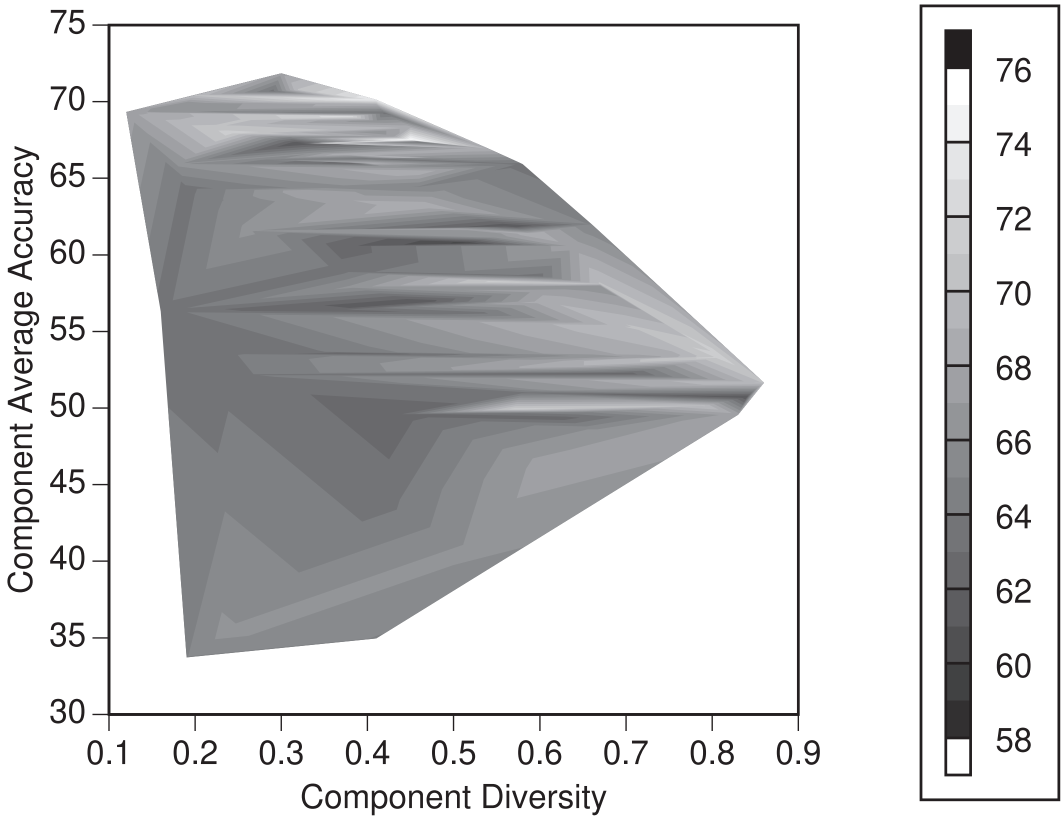

Figure B.4: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Diabetes data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

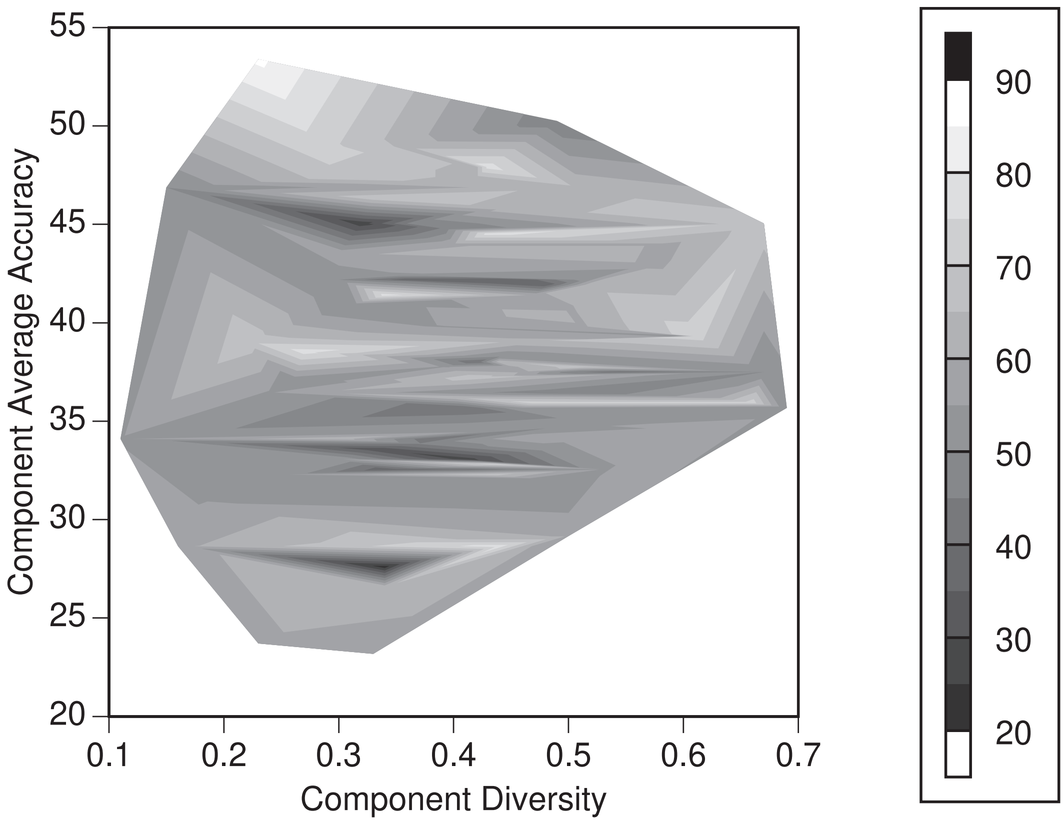

Figure B.5: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Glass Recognition data.Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

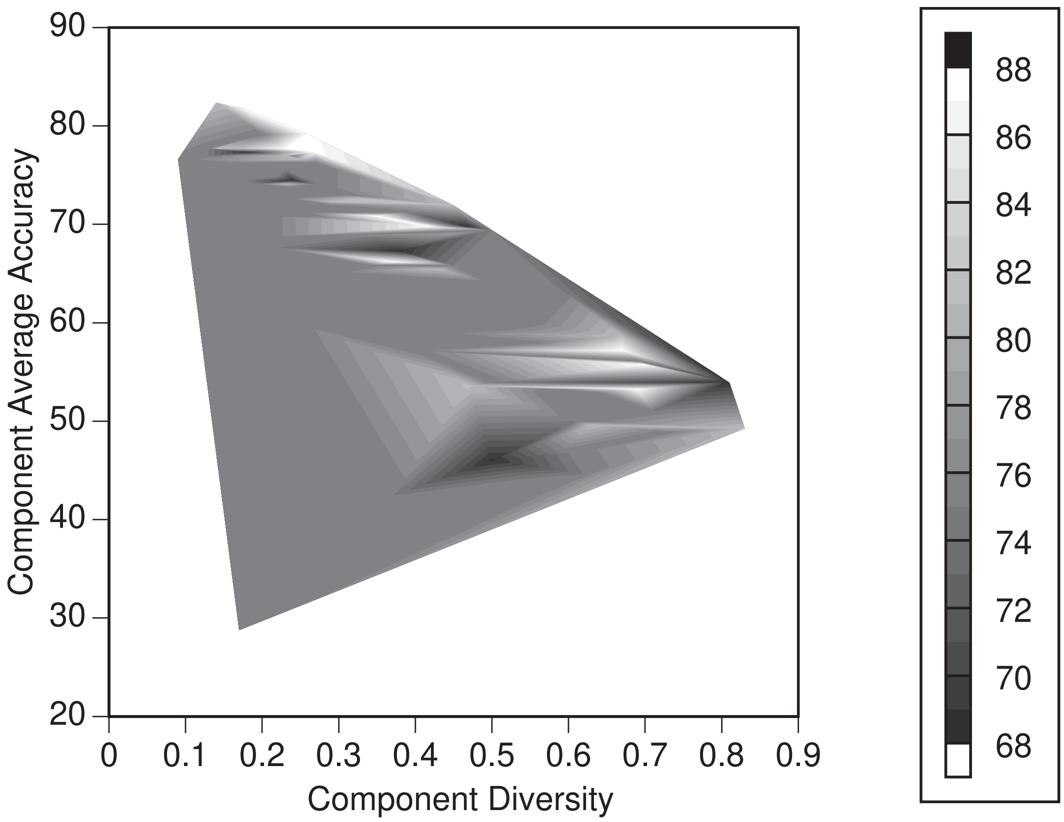

Figure B.6: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Hepatitis data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

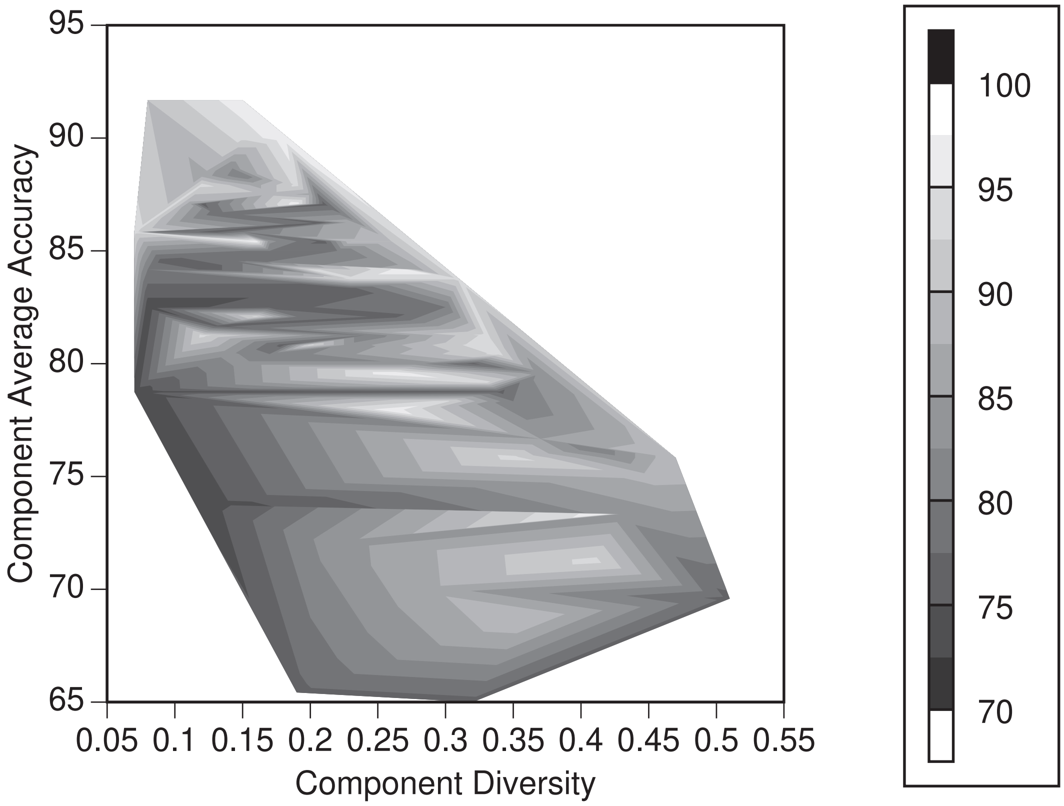

Figure B.7: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Iris data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

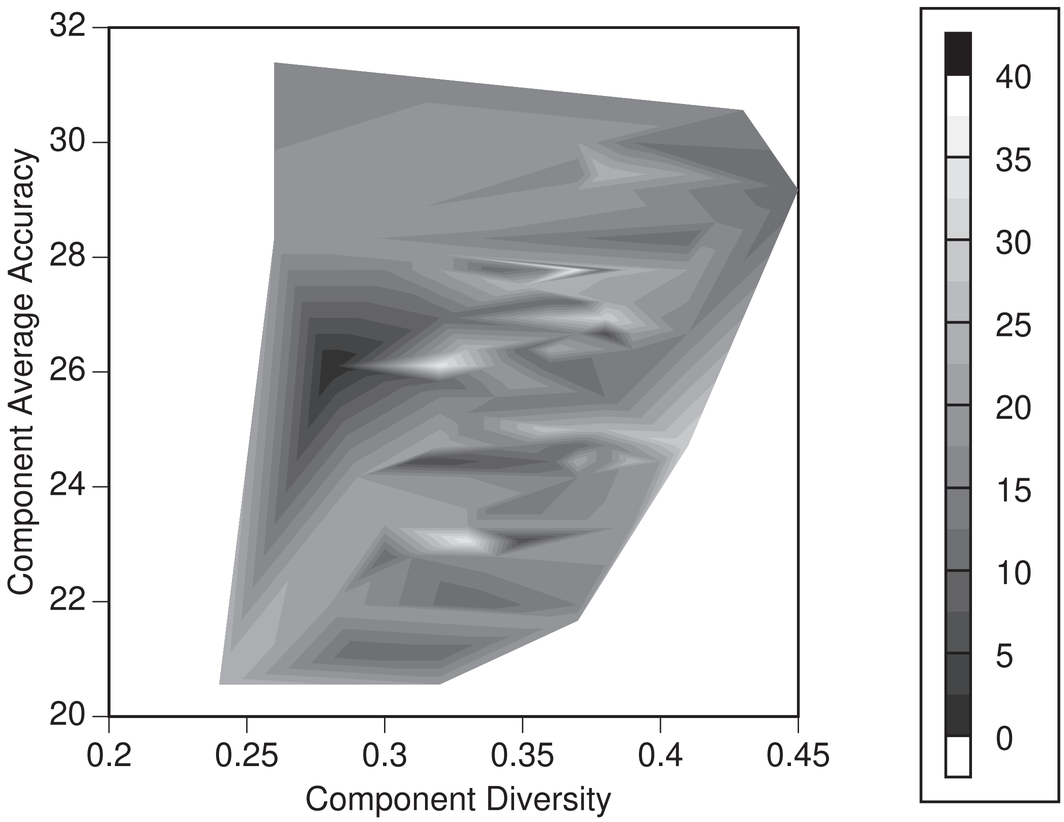

Figure B.8: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the LED-24 Digit data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

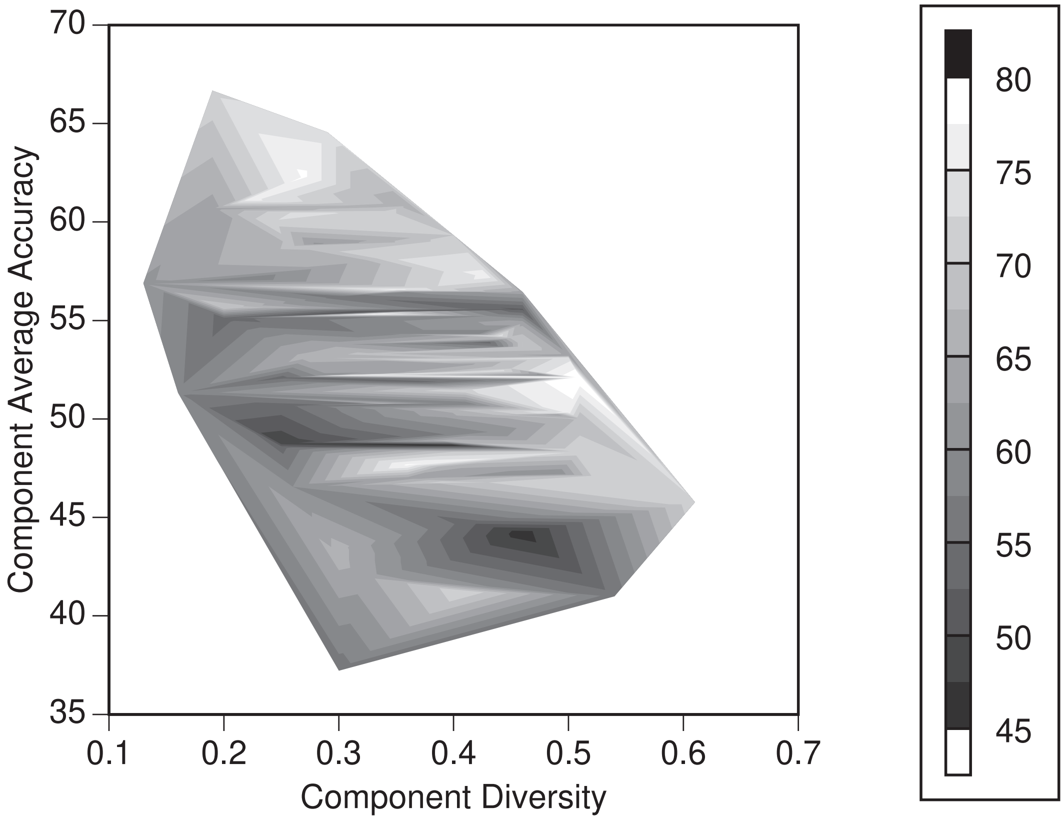

Figure B.9: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the LED-7 Digit data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

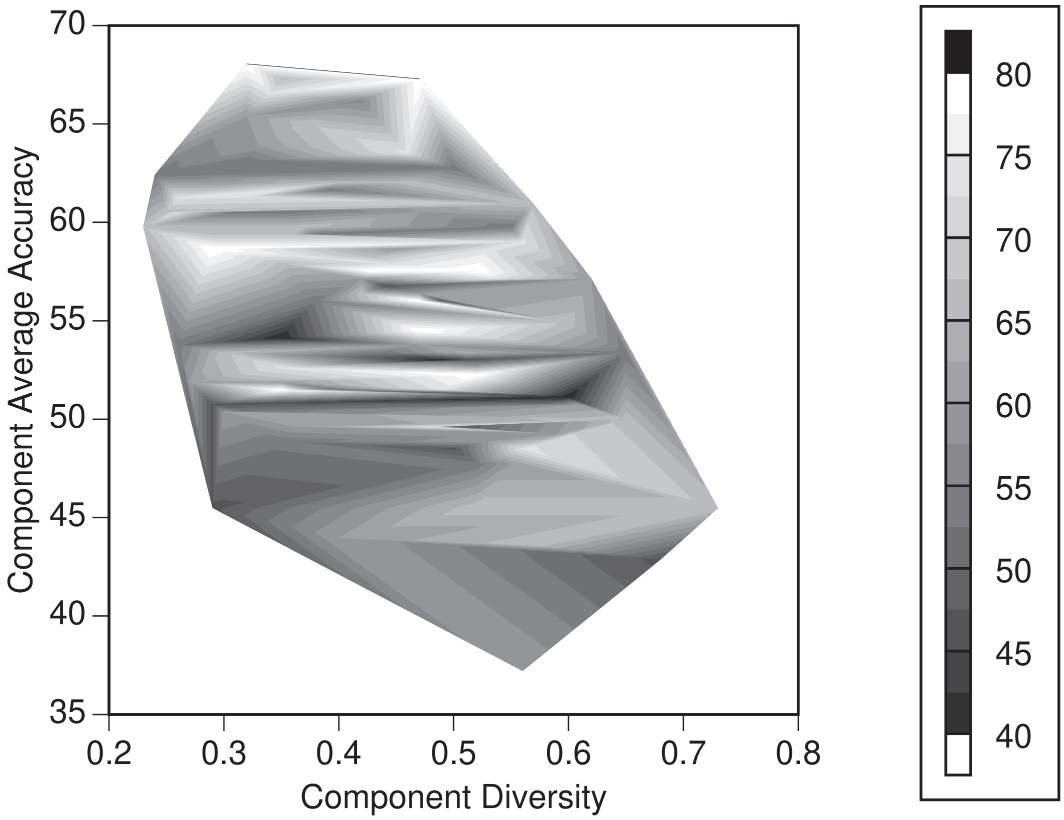

Figure B.10: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Lymphography data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

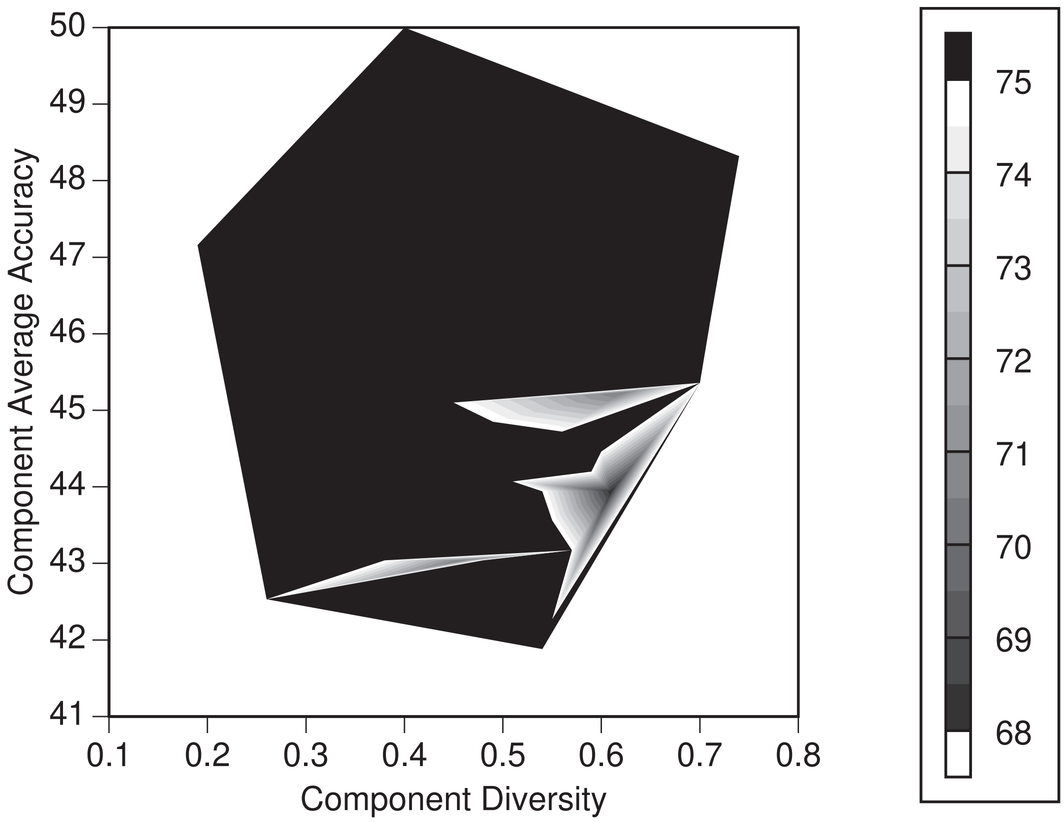

Figure B.11: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Monks-2 data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

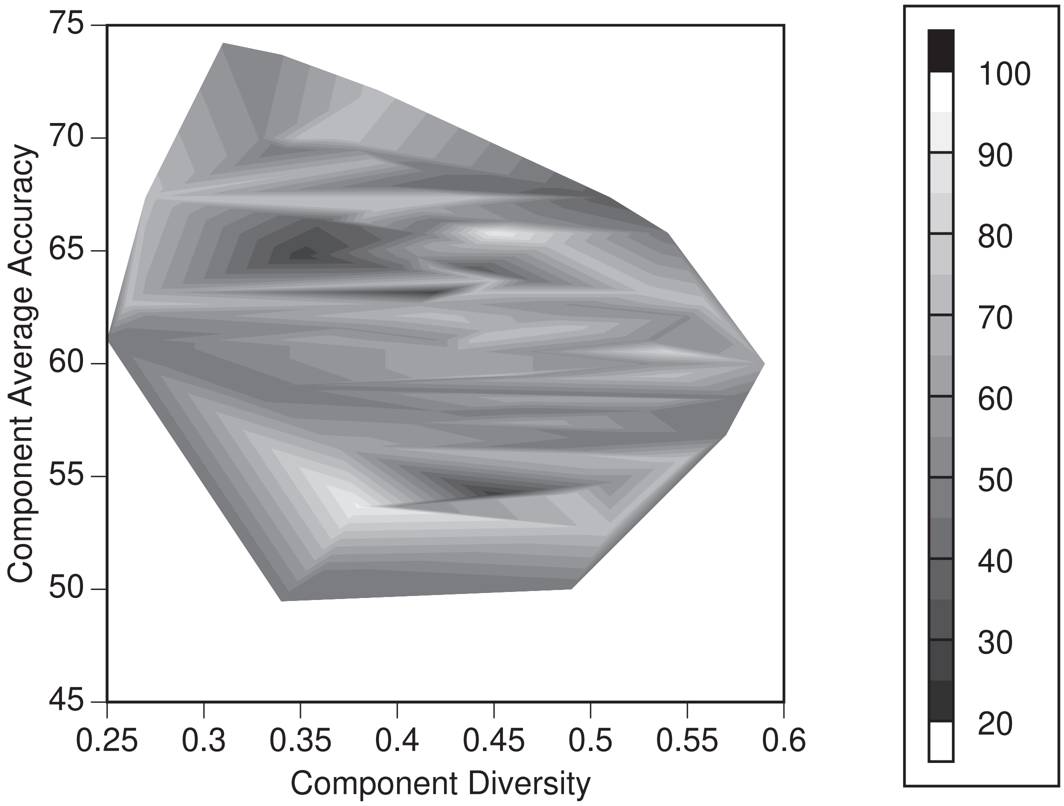

Figure B.12: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Promoter data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

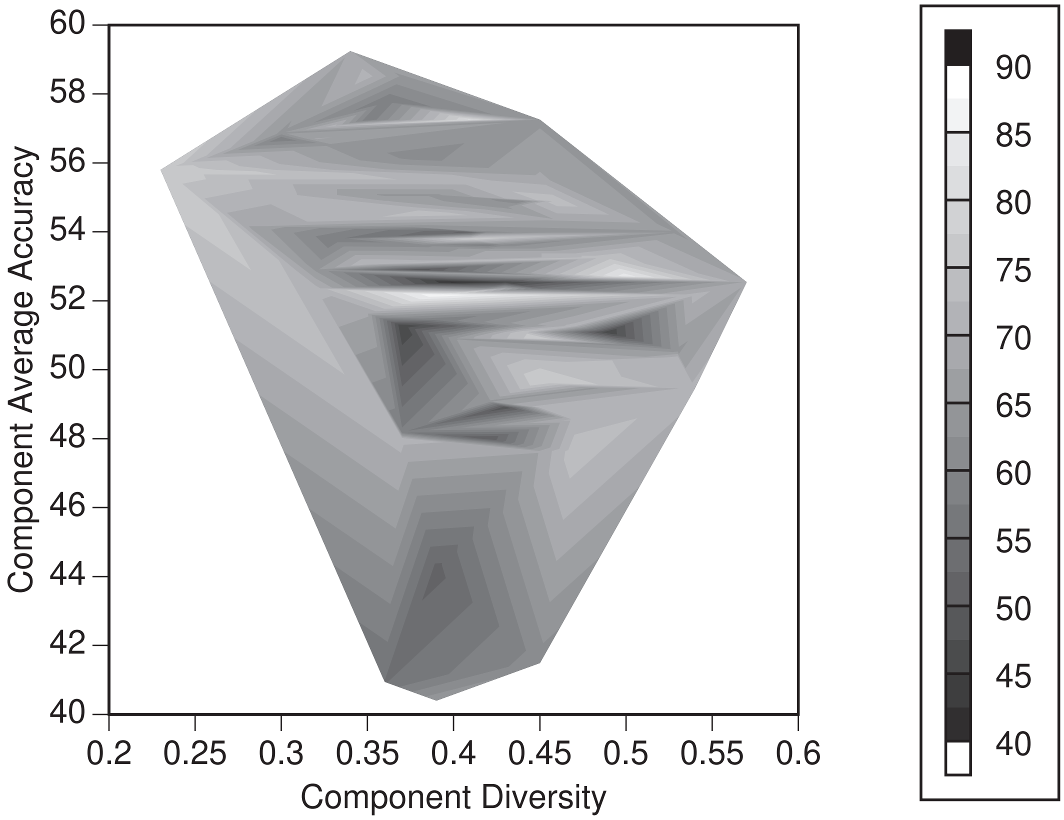

Figure B.13: Contour graph of composite accuracy plotted against component accuracy and diversity for stacked generalizers on the Soybean data. Each stacked generalizer applied ID3 as a combining algorithm and incorporated two nearest neighbor level-0 classifiers, each of which stored one randomly selected prototype per class.

APPENDIX C

SUM OF SQUARES AND CROSS-PRODUCT MATRICES

Let there be g clusters of real vectors $\mathbf{x}_{lj}$, $l = 1, 2, \ldots, g$, each of which has n_l instances indexed by $j = 1, 2, \ldots, n_l$. Let $\bar{\mathbf{x}}$ be the overall sample mean vector, and $\bar{\mathbf{x}}_l$ be the mean vector of the vectors in cluster l .

Then we can define $\mathbf{B}$ the *between cluster sum of squares and cross product matrix* as

$$\sum_{l=1}^{g} n_l(\bar{\mathbf{x}}_l - \bar{\mathbf{x}})(\bar{\mathbf{x}}_l - \bar{\mathbf{x}})'$$

We also define $\mathbf{W}$, the *pooled within cluster sum of squares and cross product matrix* as

$$\sum_{l=1}^{g} \sum_{j=1}^{n_l} (\mathbf{x}_{lj} - \bar{\mathbf{x}}_l)(\mathbf{x}_{lj} - \bar{\mathbf{x}}_l)'$$

BIBLIOGRAPHY

[Aha, 1990] Aha, D. W. 1990. *A Study of Instance-Based Algorithms for Supervised Learning Tasks: Mathematical, Empirical, and Psychological Evaluations.* Ph.D. Dissertation, Dept. of Information and Computer Science, University of California, Irvine.

[Ali and Pazzani, 1995] Ali, K.M. and Pazzani, M.J. 1995. On the link between error correlation and error reduction in decision tree ensembles. Department of Information and Computer Science Technical Report 95-38, University of Calfornia, Irvine, CA.

[Ali and Pazzani, 1996] Ali, K.M. and Pazzani, M.J. 1996. Error reduction through learning multiple descriptions. *Machine Learning* To appear. Also published as Dept. of Information and Computer Science Technical Report 95-39, University of California, Irvine, CA.

[Ali, 1996] Ali, K.M. 1996. *Learning Probabilistic Relational Concept Descriptions.* Ph.D. Dissertation, Dept. of Information and Computer Science, University of California at Irvine, Irvine, CA.

[Angluin, 1992] Angluin, D. 1992. Computational Learning Theory: Survey and Selected Bibliography. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, Victoria, B.C., Canada. Association for Computing Machinery. 351–369.

[Arrow, 1963] Arrow, K.J. 1963. *Social Choice and Individual Values.* Yale University Press, New Haven, CT.

[Ash, 1989] Ash, T. 1989. Dynamic Node Creation in Backpropagation Networks. *Connection Science* 1:365–375.

[Ashley, 1990] Ashley, K. D. 1990. *Modeling Legal Argument: Reasoning with Cases and Hypotheticals.* M.I.T. Press, Cambridge, MA.

[Atkeson *et al.*, 1996] Atkeson, C.G.; Moore, A.W.; and Schaal, S. 1996. Locally weighted learning. *Submitted to Artificial Intelligence Review.*

[Bareiss, 1989] Bareiss, E. R. 1989. *Exemplar-Based Knowledge Acquisition.* Academic Press, Boston, MA.

[Barr *et al.*, 1981] Barr, A.; Feigenbaum, E. A.; and Cohen, P. 1981. *The Handbook of Artificial Intelligence.* Addison-Wesley, Reading, MA.

[Battiti and Colla, 1994] Battiti, R. and Colla, A.M. 1994. Democracy in Neural Nets: Voting Schemes for Classification. *Neural Networks* 7:691–707.

[Benediktsson *et al.*, 1993] Benediktsson, J.A.; Sveinsson, J.R.; Swain, P.H.; and O.K.Ersoy, 1993. Parallel Consensual Neural Networks. In *Proceedings of the 1993 IEEE International Conference on Neural Networks.* IEEE Computer Society Press. 27–32.

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

[Bottou and Vapnik, 1992] Bottou, L. and Vapnik, V. 1992. Local Learning Algorithms. *Neural Computation* 4:888–900.

[Breiman *et al.*, 1984] Breiman, L.; Friedman, J.H.; Olshen, R.A.; and Stone, C.J. 1984. *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA.

[Breiman, 1992] Breiman, L. 1992. Stacked Regressions. Technical Report 367, Department of Statistics, University of California, Berkeley, CA.

[Breiman, 1994a] Breiman, L. 1994a. Bagging predictors. Technical Report 421, Department of Statistics, University of California, Berkeley, CA.

[Breiman, 1994b] Breiman, L. 1994b. NIPS*94 Tutorial, Statistics and Nets: Understanding Nonlinear Models from Their Linear Relatives. Neural Information Processing Systems, 1994, Denver, CO.

[Brodley, 1992] Brodley, C.E. 1992. Dynamic Automatic Model Selection. Technical Report 92-30, Dept. of Computer Science, University of Massachusetts, Amherst, MA.

[Brodley, 1994] Brodley, C.E. 1994. *Recursive Automatic Algorithm Selection for Inductive Learning.* Ph.D. Dissertation, Dept. of Computer Science, University of Massachusetts, Amherst, MA. (Available as Dept. of Computer Science Technical Report 96-61).

[Buntine, 1990] Buntine, W.L. 1990. *A Theory of Learning Classification Rules.* Ph.D. Dissertation, School of Computing Science, University of Technology, Sydney, Australia.

[Burrascano, 1991] Burrascano, P. 1991. Learning Vector Quantization for the Probabilistic Neural Network. *IEEE Transactions on Neural Networks* 2:458–641.

[Calinski and Harabasz, 1974] Calinski, T. and Harabasz, J. 1974. A Dendrite Method for Cluster Analysis. *Communications in Statistics* 3:1–27.

[Callan *et al.*, 1991] Callan, J. P.; Fawcett, T. E.; and Rissland, E. L. 1991. CABOT: An Adaptive Approach to Case-Based Search. In *Proceedings, 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia. International Joint Conferences on Artificial Intelligence, Inc. 803–808.

[Callan, 1993] Callan, J.P. 1993. *Knowledge-Based Feature Generation for Inductive Learning.* Ph.D. Dissertation, Department of Computer Science, University of Massachusetts, Amherst, MA.

[Cameron-Jones, 1995] Cameron-Jones, M. 1995. Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. In *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence.* World Scientific. 99–106.

[Chan and Stolfo, 1993a] Chan, P.K. and Stolfo, S.J. 1993a. Experiments on multistrategy learning by meta-learning. In *Proceedings of the Second International Conference on Information and Knowledge Management.* 314–323.

[Chan and Stolfo, 1993b] Chan, P.K. and Stolfo, S.J. 1993b. Toward Parallel and Distributed Learning by Meta-Learning. In *Working Notes, AAAI Workshop on Knowledge Discovery in Databases*, San Mateo, CA. AAAI Press/MIT Press. 227–240.

[Chan and Stolfo, 1995] Chan, P.K. and Stolfo, S.J. 1995. A Comparative Evaluation of Voting and Meta-Learning on Partitioned Data. In *Proceedings of the Twelfth International Conference on Machine Learning*, San Mateo, CA. Morgan Kaufmann. 90–98.

[Chang, 1974] Chang, C. L. 1974. Finding Prototypes for Nearest Neighbor Classifiers. *IEEE Transactions on Computers* c-23:1179–1184.

[Chatfield, 1984] Chatfield, C. 1984. *The Analysis of Time Series: An Introduction.* Smith, Bristol, UK, third edition.

[Cherkauer and Shavlik, 1996] Cherkauer, K.J. and Shavlik, J.W. 1996. Growing simpler decision trees to facilitate knowledge discovery. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.* AAAI Press, San Mateo, CA. 315–318.

[Clemen, 1989] Clemen, R.T. 1989. Combining Forecasts: A Review and Annotated Bibliography. *International Journal of Forecasting* 5:559–583.

[Cognitive Systems, Inc., 1990] Cognitive Systems, Inc., 1990. Case-Based Retrieval Shell, User's Manual V. 3.17. Cognitive Systems, Inc.

[Cognitive Systems, Inc., 1992] Cognitive Systems, Inc., 1992. ReMind: Case-based Reasoning Development Shell.

[Cooper *et al.*, 1982] Cooper, L.N.; Elbaum, C.; and Reilly, D.L. 1982. Self Organizing General Pattern Class Separator and Identifier. U.S. Patent 4,326,259.

[Cost and Salzberg, 1993] Cost, S. and Salzberg, S. 1993. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning* 10:57–78.

[Cover and Hart, 1967] Cover, T. M. and Hart, P. E. 1967. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory* IT-13:21–27.

[Dasarathy, 1980] Dasarathy, B. V. 1980. Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2:67–71.

[Dasarathy, 1991] Dasarathy, B. V. 1991. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA.

[Datta and Kibler, 1995] Datta, P. and Kibler, D. 1995. Learning prototypical concept descriptions. In *Proceedings of the Twelfth International Conference on Machine Learning.* Morgan Kaufmann, San Mateo, CA. 158–166.

[de la Maza, 1991] Maza, M.de la 1991. A Prototype Based Symbolic Concept Learning System. In *Proceedings of the Eighth International Workshop on Machine Learning*, San Mateo, CA. Morgan Kaufmann. 41–45.

[Dempster, 1968] Dempster, A.P. 1968. A generalization of bayesian inference. *J. Roy. Stat. Soc. B* 30:205–247.

[Devijver and Kittler, 1980] Devijver, P. A. and Kittler, J. 1980. On the Edited Nearest Neighbor Rule. In *Proceedings of the 5th International Conference on Pattern Recognition*, Los Alamitos, CA. The Institute of Electrical and Electronics Engineers. 72–80.

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

[Drucker and Cortes, 1996] Drucker, H. and Cortes, C. 1996. Boosting Decision Trees. In Touretzky, D.S.; Mozer, M.C.; and Hasselmo, M.E., editors, *Advances in Neural Information Processing Systems, 8*. MIT Press, Las Vegas, NV. 479–485.

[Drucker *et al.*, 1994a] Drucker, H.; Cortes, C.; Jackel, L.D.; LeCun, Y.; and Vapnik, V. 1994a. Boosting and other ensemble methods. *Neural Computation* 6(6):1289–1301.

[Drucker *et al.*, 1994b] Drucker, H.; Cortes, C.; Jackel, L.D.; LeCun, Y.; and Vapnik, V. 1994b. Boosting and other machine learning methods. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA. 53–61.

[Drucker, 1996] Drucker, H. 1996. Fast Decision Tree Ensembles for Optical Character Recognition. In *Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*. University of Nevada, Las Vegas, NV.

[Duda and Hart, 1973] Duda, R. O. and Hart, P. E. 1973. *Pattern Classification and Scene Analysis*. John Wiley, New York.

[Edelman, 1995] Edelman, S. 1995. Representation, Similarity, and the Chorus of Prototypes. *Minds and Machines* 5:45–68.

[Efron, 1979] Efron, B. 1979. Computers and the Theory of Statistics: Thinking the Unthinkable. *SIAM Review* 21:460–480.

[English and Gotesman, 1995] English, T.M. and Gotesman, M. 1995. Stacked generalization and fitness ranking in evolutionary algorithms. In *Evolutionary Programming*, volume IV. MIT Press. 205–218.

[English, 1996] English, T.M. 1996. Stacked generalization and simulated evolution. *BioSystems (to appear)*.

[Fahlman and Lebiere, 1990] Fahlman, S.E. and Lebiere, C. 1990. The Cascade Correlation Architecture. *Advances in Neural Information Processing Systems* 2:524–532.

[Feng *et al.*, 1993] Feng, C.; King, R.; and Sutherland, A. 1993. Statlog: Compaarison of machine learning, statistical and neural network classification algorithms. Technical report, The Turing Institute.

[Fisher, 1936] Fisher, R. A. 1936. The use of multiple measurements in taxonomic problems. *Annual Eugenics* 7:179–188.

[Frean, 1990] Frean, M. 1990. The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation* 2:198–209.

[Freund and Schapire, 1995] Freund, Y. and Schapire, R.E. 1995. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*. Springer Verlag, Barcelona, Spain. 23–37.

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

[Gates, 1972] Gates, G. W. 1972. The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory* IT-18, No. 3:431–433.

[Geman *et al.*, 1996] Geman, D.; Amit, Y.; and Wilder, K. 1996. Joint induction of shape features and tree classifiers. Technical report, Dept of Mathematics, University of Massachusetts, Amherst, MA.

[Ginsberg, 1993] Ginsberg, M. 1993. *Essentials of Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA.

[Goel, 1989] Goel, A. 1989. *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving*. Ph.D. Dissertation, Dept. of Computer and Information Science, The Ohio State University, Columbus, OH.

[Golding and Rosenbloom, 1991] Golding, A. R. and Rosenbloom, P. S. 1991. Improving Rule-Based Systems through Case-Based Reasoning. In *Ninth National Conference on Artificial Intelligence*, Anaheim, CA. American Association for Artificial Intelligence. 22–27.

[Guttman *et al.*, 1971] Guttman, I.; Wilks, S.S.; and Hunter, J.S. 1971. *Introductory Engineering Statistics*. John Wiley and Sons, New York, NY, second edition.

[Hansen and Salamon, 1990] Hansen, L.K. and Salamon, P. 1990. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12:993–1001.

[Hansen *et al.*, 1953] Hansen, M.H.; Hurwitz, W.N.; and Madow, W.G. 1953. *Sample Survey Methods and Theory*, volume I. John Wiley and Sons, New York, NY.

[Hart, 1968] Hart, P. E. 1968. The Condensed Nearest Neighbor Rule. *IEEE Transactions on Information Theory (Corresp.)* IT-14:515–516.

[Hashem, 1993] Hashem, S. 1993. *Optimal Linear Combinations of Neural Networks.* Ph.D. Dissertation, Purdue University, School of Industrial Engineering, West Lafayette, IN.

[Heath *et al.*, 1996] Heath, D.; Kasif, S.; and Salzberg, S. 1996. Committees of decision trees. In Gorayska, B. and Mey, J., editors, *Cognitive Technology: In Search of a Human Interface.* Elsevier Science, Amsterdam, The Netherlands. 305–317.

[Hertz *et al.*, 1991] Hertz, J.; Krogh, A.; and Palmer, R. G. 1991. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA.

[Ho *et al.*, 1994] Ho, T.K.; Hull, J.J.; and Srihari, S.N. 1994. Decision Combination in Multiple Classifier Systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16:66–75.

[Holte, 1993] Holte, R. C. 1993. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning* 11:63–90.

[Hutchinson, 1993] Hutchinson, J. M. 1993. *A Radial Basis Function Approach to Financial Time Series Analysis.* Ph.D. Dissertation, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

[Intrator and Edelman, 1996] Intrator, N. and Edelman, S. 1996. Making a low-dimensional representation suitable for diverse tasks. *Connection Science, to appear.*

[Jacobs *et al.*, 1991] Jacobs, R. A.; Jordan, M. I.; and Barto, A. G. 1991. Task Decomposition through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *Cognitive Science* 15:219–250.

[Johnson and Wichern, 1992] Johnson, R.A. and Wichern, D.W. 1992. *Applied Multivariate Statistical Analysis.* Prentice-Hall, Englewood Cliffs, NJ.

[Jordan and Jacobs, 1993] Jordan, M.I. and Jacobs, R.A. 1993. Hierarchical Mixtures of Experts and the EM Algorithm. Technical Report 1440, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, MA.

[Kearns, 1988] Kearns, M. 1988. Thoughts on hypothesis boosting. (unpublished manuscript).

[Kemeny, 1953] Kemeny, J.G. 1953. The use of simplicity in induction. *The Philosophical Review* LXII(3):391–408.

[Kibler and Aha, 1988] Kibler, D. and Aha, D. W. 1988. Comparing Instance-Averaging with Instance-Filtering Learning Algorithms. In *Proceedings of the Third European Working Session on Learning,* Glasgow. Pitman. 63–80.

[Kohonen *et al.*, 1988] Kohonen, T.; Barna, G.; and Chrisley, R. 1988. Statistical Pattern Recognition with Neural Networks: Benchmarking Studies. In *IEEE International Conference on Neural Networks,* San Diego, CA. IEEE. I61–I68.

[Kohonen, 1989] Kohonen, T. 1989. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, third edition.

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

[Koton, 1988] Koton, P. A. 1988. *Using Experience in Learning and Problem Solving.* Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA.

[Krogh and Vedelsby, 1995] Krogh, A. and Vedelsby, J. 1995. Neural Network Ensembles, Cross Validation and Active Learning. In Tesauro, G.; Touretzky, D.; and Leen, T., editors, *Advances in Neural Information Processing Systems, 7.* MIT Press. 231–238.

[Krzanowski, 1988] Krzanowski, W. J. 1988. *Principles of Multivariate Analysis.* Clarendon Press, Oxford, UK.

[Kurtzberg, 1987] Kurtzberg, J. M. 1987. Feature Analysis for Symbol Recognition by Elastic Matching. *International Business Machines Journal of Research and Development* 31:91–95.

[Kwok and Carter, 1990] Kwok, S.W. and Carter, C. 1990. Multiple decision trees. In Schachter, R.D.; Levitt, T.S.; Kanal, L.N.; and Lemmer, J.F., editors, *Uncertainty in Artificial Intelligence*, volume 4. Elsevier Science (North Holland). 327–335.

[Lakoff, 1987] Lakoff, G. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* University of Chicago Press, Chicago.

[LeCun *et al.*, 1989] LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; and Jackel, L.D. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1:541–551.

[Lesser *et al.*, 1995] Lesser, V. R.; Nawab, S. H.; and Klassner, F. I. 1995. IPUS: An Architecture for the Integrated Processing and Understanding of Signals. *Artificial Intelligence Journal* 77:129–171.

[Levi, 1949] Levi, E.H. 1949. *An Introduction to Legal Reasoning.* University of Chicago, Chicago, IL.

[Littlestone and Warmuth, 1989] Littlestone, N. and Warmuth, M. 1989. The Weighted Majority Algorithm. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, Washington, DC. IEEE Computer Society Press. 256–261.

[Littlestone, 1987] Littlestone, N. 1987. Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, Washington, DC. IEEE Computer Society Press. 68–77.

[Mackey and Glass, 1977] Mackey, M.C. and Glass, L. 1977. Oscillation and chaos in physiological control systems. *Science* 197:287.

[Maclin and Shavlik, 1995] Maclin, R. and Shavlik, J.W. 1995. Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks. In Mellish, C.S., editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA. 524–530.

[Mandler and Schurmann, 1988] Mandler, E. and Schurmann, J. 1988. Combining the classification results of independent classifiers based on the dempster/schafer theory of evidence. In Gelsema, E.S. and Kanal, L.N., editors, *Pattern Recognition and Artificial Intelligence*, volume 7 of *Machine Intelligence and Pattern Recognition*. North-Holland, Amsterdam, The Netherlands. 381–393.

[Mangasarian and Wolberg, 1990] Mangasarian, O.L. and Wolberg, W.H. 1990. Cancer Diagnosis via Linear Programming. *SIAM News* 23:1–18.

[Mani, 1991] Mani, G. 1991. Lowering variance of decisions by using artificial neural networks. *Neural Computation* 3:484–486.

[Marchand *et al.*, 1990] Marchand, M.; Golea, M.; and Ruján, P. 1990. A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons. *Europhysics Letters* 11:487–492.

[McCarty and Sridharan, 1982] McCarty, L. T. and Sridharan, N. S. 1982. A Computational Theory of Legal Argument. Technical Report LRP-TR-13, Laboratory for Computer Science Research, Rutgers University.

[Mezard and Nadal, 1989] Mezard, M. and Nadal, J.P. 1989. Learning in Feedforward Layered Networks: The Tiling Algorithm. *Journal of Physics A* 22:2191–2204.

[Michalski, 1994] Michalski, R.S. 1994. Inferential Theory of Learning: A Multistrategy Approach. In Michalski, R. and Tecuci, G., editors, *Machine Learning Vol. IV*. Morgan Kaufmann, San Francisco, CA. 3–61.

[Milligan and Cooper, 1985] Milligan, G. W. and Cooper, M. C. 1985. An Examination of Procedures for Determining the Number of Clusters in a Data set. *Psychometrika* 50:159–179.

[Minsky, 1965] Minsky, M. 1965. Steps Toward Artificial Intelligence. In R. D. Luce, R. R. Bush and Galanter, E., editors, *Readings in Mathematical Psychology (originally published in the Proceedings of the Institute for Radio and Electronics, 1961, vol. 49, pp. 8-30)*. John Wiley, New York.

[Moody and Darken, 1989] Moody, J. and Darken, C. 1989. Fast Learning in Networks of Locally-Tuned Processing Units. Technical Report YALEU/DCS/RR-654, Yale University Department of Computer Science, New Haven, CT.

[Munkres, 1975] Munkres, J.R. 1975. *Topology: A First Course*. Prentice-Hall, Englewood Cliffs, NJ.

[Murphy and Aha, 1994] Murphy, P. M. and Aha, D. W. 1994. University of California at Irvine Repository of Machine Learning Databases. For information contact ml-repository@ics.uci.edu.

[Nilsson, 1990] Nilsson, N. J. 1990. *The Mathematical Foundations of Learning Machines.* Morgan Kaufmann, San Mateo, CA.

[Nowlan, 1990] Nowlan, S.J. 1990. Competing Experts: An Experimental Investigation of Associative Mixture Models. Connectionist Research Group Technical Report CRG-TR-90-5, Dept. of Computer Science, University of Toronto, Toronto, Ontario.

[Oliver and Hand, 1995] Oliver, J.J. and Hand, D.J. 1995. On pruning and averaging decision trees. In *Proceedings of the Twelfth International Conference on Machine Learning.* Morgan Kaufmann, Tahoe City, CA. 430–437.

[Opitz and Shavlik, 1995] Opitz, D.W. and Shavlik, J.W. 1995. Generating Accurate and Diverse Members of a Neural-Network Ensemble. In Touretzky, D.S.; Mozer, M.C.; and Hasselmo, M.E., editors, *Advances in Neural Information Processing Systems, 8.* MIT Press, Cambridge, MA. 535–541.

[Perrone, 1993] Perrone, M.P. 1993. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization.* Ph.D. Dissertation, Dept. of Computer Science, Brown University, Providence, RI.

[Plutowski *et al.*, 1994] Plutowski, M.; Cottrell, G.; and White, H. 1994. Learning Mackey-Glass from 25 examples, Plus or Minus 2. In Cowan, J.D.; Tesauro, G.; and Alspector, J., editors, *Advances in Neural Information Processing Systems, 6*, San Mateo, CA. Morgan Kaufmann. 1135–1142.

[Poggio and Girosi, 1990] Poggio, T. and Girosi, F. 1990. Networks for Approximation and Learning. *Proceedings of the IEEE* 79:1481–1497.

[Quinlan, 1986] Quinlan, J. R. 1986. Induction of Decision Trees. *Machine Learning* 1:81–106.

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

[Quinlan, 1996] Quinlan, J.R. 1996. Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence.* AAAI Press/MIT Press, Menlo Park, CA. 725–730.

[Rao *et al.*, 1994] Rao, N.S.V.; Oblow, E.M.; Glover, C.W.; and Liepins, G.E. 1994. N-Learners Problem: Fusion of Concepts. *IEEE Transactions on Systems, Man, and Cybernetics* 24:319–326.

[Ripley, 1996] Ripley, B.D. 1996. *Pattern Recognition and Neural Networks.* Cambridge, Cambridge, England.

[Rissland and Ashley, 1986] Rissland, E.L. and Ashley, K.D. 1986. Hypotheticals as Heuristic Device. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA. American Association for Artificial Intelligence. 289–297.

[Rissland and Skalak, 1991] Rissland, E. L. and Skalak, D. B. 1991. CABARET: Rule Interpretation in a Hybrid Architecture. *International Journal of Man-Machine Studies* 34:839–887.

[Rissland *et al.*, 1984] Rissland, E.L.; Valcarce, E.M.; and Ashley, K.D. 1984. Explaining and Arguing with Examples. In *AAAI-84, Proceedings of the National Conference on Artificial Intelligence*, Austin, TX. American Association for Artificial Intelligence.

[Rissland *et al.*, 1993] Rissland, E.L.; Daniels, J.J.; Rubinstein, Z.B.; and Skalak, D.B. 1993. Case-Based Diagnostic Analysis in a Blackboard Architecture. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, San Mateo, CA. AAAI Press/MIT Press. 66–72.

[Rissland *et al.*, 1996] Rissland, E.L.; Skalak, D.B.; and Friedman, M.T. 1996. Bankxx: Supporting legal arguments through heuristic retrieval. *Artificial Intelligence and Law* 4:1–71.

[Rissland, 1977] Rissland, E. L. 1977. *Epistemology, Representation, Understanding and Interactive Exploration of Mathematical Theories.* Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

[Rissland, 1978a] Rissland, E. L. 1978a. Understanding Understanding Mathematics. *Cognitive Science* 2:361–383.

[Rissland, 1978b] Rissland, E.L. 1978b. *The Structure of Mathematical Knowledge.* Ph.D. Dissertation, Massachusetts Institute of Technology.

[Rissland, 1981] Rissland, E. L. 1981. Constrained Example Generation. Dept. of Computer Science Technical Report 81-24, University of Massachusetts, Amherst, MA.

[Rissland, 1985] Rissland, E. L. 1985. Argument Moves and Hypotheticals. In Walter, C., editor, *Computing Power and Legal Reasoning.* West Publishing Co., St. Paul, MN.

[Rissland, 1989] Rissland, E. L. 1989. Case-Based Reasoning, Introduction to the Proceedings. In *Proceedings: Case-Based Reasoning Workshop*, Pensacola Beach, FL. Morgan Kaufmann.

[Rissland, 1990] Rissland, E. L. 1990. Dimension-based Analysis of Hypotheticals from Supreme Court Oral Argument. *Proceedings of the Second International Conference on AI and Law* 111–120.

[Ritter *et al.*, 1975] Ritter, G. L.; Woodruff, H. B.; Lowry, S. R.; and Isenhour, T. L. 1975. An Algorithm for a Selective Nearest Neighbor Decision Rule. *IEEE Transactions on Information Theory* IT-21:665–669.

[Rosch and Mervis, 1975] Rosch, E. and Mervis, C. B. 1975. Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology* 7:573–605.

[Rosenblatt, 1962] Rosenblatt, F. 1962. *Principles of Neurodynamics.* Spartan Books.

[Salton, 1989] Salton, G. 1989. *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer.* Addison-Wesley, Reading, MA.

[Salzberg, 1991] Salzberg, S. 1991. A Nearest Hyperrectangle Learning Method. *Machine Learning* 6:251–276.

[Sanders, 1994] Sanders, K. E. 1994. *CHIRON: planning in an open-textured domain*. Ph.D. Dissertation, Computer Science Department, Brown University, Providence, RI.

[Schaffer, 1994] Schaffer, C. 1994. Cross-Validation, Stacking and Bi-Level Stacking: Meta-Methods for Classification Learning. In Cheeseman, P. and Oldford, R.W., editors, *Selecting Models from Data: Artificial Intelligence and Statistics IV*. Springer Verlag, New York, NY. 51–59.

[Schapire, 1990] Schapire, R.E. 1990. The Strength of Weak Learnability. *Machine Learning* 5:197–227.

[Sejnowski and Rosenberg, 1987] Sejnowski, T.J. and Rosenberg, C.R. 1987. Parallel networks that learn to pronounce english text. *Complex Systems* 1:145–168.

[Selfridge, 1959] Selfridge, O. G. 1959. Pandemonium: A Paradigm for Learning. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, Teddington, England. National Physical Laboratory, H.M. Stationery Office, London. 511–529.

[Seung *et al.*, 1992] Seung, H.S.; Opper, M.; and Sompolinsky, H. 1992. Query by committee. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*. ACM. 287–294.

[Shafer, 1976] Shafer, G. 1976. *A Mathematical Theory of Evidence*. Princeton University, Princeton, NJ.

[Skalak and Rissland, 1992] Skalak, D. B. and Rissland, E. L. 1992. Arguments and Cases: An Inevitable Intertwining. *Artificial Intelligence and Law: An International Journal* 1:3–48.

[Skalak, 1990] Skalak, D.B. 1990. An Internal Contradiction of Case-Based Reasoning. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Lawrence Erlbaum Associates. 109–116.

[Skalak, 1992] Skalak, D. B. 1992. Representing Cases as Knowledge Sources that Apply Local Similarity Metrics. In *The 14th Annual Conference of the Cognitive Science Society*, Bloomington, Indiana. Lawrence Erlbaum. 325–330.

[Skalak, 1993] Skalak, D. B. 1993. Using a Genetic Algorithm to Learn Prototypes for Case Retrieval and Classification. In *Proceedings of the AAAI-93 Case-Based Reasoning Workshop (Technical Report WS-93-01)*, Washington, D.C. American Association for Artificial Intelligence, Menlo Park, CA.

[Skalak, 1994] Skalak, D. B. 1994. Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ. Morgan Kaufmann. 293–301.

[Skalak, 1995] Skalak, D.B. 1995. Prototype selection for composite nearest neighbor classifiers. Technical Report 95-74, Department of Computer Science, University of Massachusetts, Amherst, MA.

[Smith and Medin, 1981] Smith, E. E. and Medin, D. L. 1981. *Categories and Concepts*. Harvard, Cambridge, MA.

[Stanfill and Waltz, 1986] Stanfill, C. and Waltz, D. 1986. Toward Memory-Based Reasoning. *Communications of the ACM* 29:1213–1228.

[Sutton and Whitehead, 1993] Sutton, R. S. and Whitehead, S. D. 1993. Online Learning with Random Representations. In *Proceedings of the Tenth International Machine Learning Conference*, Amherst, MA. Morgan Kaufmann, San Mateo, CA. 314–321.

[Swonger, 1972] Swonger, C.W. 1972. Sample Set Condensation for a Condensed Nearest Neighbor Decision Rule for Pattern Recognition. In Watanabe, S., editor, *Frontiers of Pattern Recognition*. Academic Press, New York, NY. 511–519.

[Sycara, 1987] Sycara, K. P. 1987. *Resolving Adversarial Conflicts: An Approach Integrating Case- Based and Analytic Methods*. Ph.D. Dissertation, School of Information and Computer Science, Georgia Institute of Technology.

[Tan and Schlimmer, 1990] Tan, M. and Schlimmer, J. C. 1990. Two Case Studies in Cost-Sensitive Concept Acquisition. In *Proceedings, Eighth National Conference on Artificial Intelligence*, Boston, MA. AAAI Press, Menlo Park, CA. 854–860.

[Thodberg, 1995] Thodberg, H.H. 1995. A review of bayesian neural networks with an application to near infared spectroscopy. Technical report, The Danish Meat Research Institute.

[Thompson, 1992] Thompson, S.K. 1992. *Sampling*. John Wiley and Sons, New York, NY.

[Thrun *et al.*, 1991] Thrun, S.B.; Bala, J.; Bloedorn, E.; Bratko, I.; Cestnik, B.; Cheng, J.; Jong, K. De; Dzeroski, S.; Fahlman, S.E.; Fisher, D.; Hamann, R.; Kaufman, K.; Keller, S.; Kononenko, I.; Kruziger, J.; Michalski, R.S.; Mitchell, T.; Pachowicz, P.; Vafaie, Y. Reich H.; Welde, W. Vande; Wenzel, W.; Wnek, J.; and Zhang, J. 1991. The MONK's Problems - A Performance Comparison of Different Learning algorithms. Dept. of Computer Science Technical Report CS-CMU-91-197, Carnegie Mellon University.

[Tomek, 1976] Tomek, I. 1976. An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6:448–452.

[Tumer and Ghosh, 1996] Tumer, K. and Ghosh, J. 1996. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition* 29(2):341–348.

[Utgoff and Clouse, 1996] Utgoff, P.E. and Clouse, J.A. 1996. A Kolmogorov-Smirnoff Metric for Decision Tree Induction. Technical Report 93-3, Dept. of Computer Science, University of Massachusetts, Amherst, MA.

[Utgoff, 1989] Utgoff, P.E. 1989. Perceptron Trees: A Case Study in Hybrid Concept Representations. *Connection Science* 1:377–391.

[Valiant, 1984] Valiant, L.G. 1984. A Theory of the Learnable. *Communications of the ACM* 27:1134–1142.

[Voisin and Devijver, 1987] Voisin, J. and Devijver, P. A. 1987. An application of the Multiedit-Condensing technique to the reference selection problem in a print recognition system. *Pattern Recognition* 5:465–474.

[Vossos *et al.*, 1991] Vossos, G.; Zeleznikow, J.; Dillon, T.; and Vossos, V. 1991. An Example of Integrating Legal Case Based Reasoning with Object-Oriented Rule-Based Systems: IKBALS II. In *Proceedings of the Third International Conference on Artificial Intelligence and Law*, Oxford, England. ACM. 31–41.

[Walker, 1992] Walker, R. 1992. *An Expert System Architecture for Heterogeneous Domains.* Ph.D. Dissertation, Vrije Universiteit te Amsterdam.

[Wasserman, 1993] Wasserman, P. D. 1993. *Advanced Methods in Neural Computing.* Van Nostrand Reinhold, New York.

[Weiss and Kapouleas, 1989] Weiss, S. M. and Kapouleas, I. 1989. An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. In *11th International Joint Conference on Artificial Intelligence*, Detroit, MI. International Joint Conferences on Artificial Intelligence. 781–787.

[Wettschereck and Dietterich, 1992] Wettschereck, D. and Dietterich, T. 1992. Improving the Performance of Radial Basis Function Networks by Learning Center Locations. In Moody, J.E.; Hanson, S.J.; and Lippmann, R.P., editors, *Advances in Neural Information Processing Systems, 4*, San Mateo, CA. Morgan Kaufmann. 1133–1140.

[Wettschereck, 1994] Wettschereck, D. 1994. *A Study of Distance-Based Machine Learning Algorithms.* Ph.D. Dissertation, Dept. of Computer Science, Oregon State University, Corvallis, Oregon.

[Wilson, 1972] Wilson, D. 1972. Asymptotic Properties of Nearest Neighbor Rules using Edited Data. *Institute of Electrical and Electronic Engineers Transactions on Systems, Man and Cybernetics* 2:408–421.

[Wittgenstein, 1953] Wittgenstein, L. 1953. *Philosophical Investigations.* Macmillan, New York, NY.

[Wolpert, 1992] Wolpert, D. 1992. Stacked Generalization. *Neural Networks* 5:241–259.

[Wolpert, 1993] Wolpert, D. 1993. Combining Generalizers using Partitions of the Learning Set. In Nadel, L. and Stein, D., editors, *1992 Lectures in Complex Systems.* Addison-Wesley, Reading, MA.

[Wonnacott and Wonnacott, 1972] Wonnacott, T.H. and Wonnacott, R.J. 1972. *Introductory Statistics.* John Wiley and Sons, New York, NY, second edition.

[Xu *et al.*, 1992] Xu, L.; Kryzak, A.; and Suen, C.Y. 1992. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics* 22:418–435.

[Zhang *et al.*, 1992] Zhang, X.; Mersirov, J.P.; and D.L.Waltz, 1992. A Hybrid System for Protein Secondary Structure Prediction. *Journal of Molecular Biology* 225:1049–1063.

[Zhang, 1992] Zhang, J. 1992. Selecting Typical Instances in Instance-Based Learning. In *Proceedings of the Ninth International Machine Learning Workshop*, Aberdeen, Scotland. Morgan Kaufmann, San Mateo, CA. 470–479.

[Zheng, 1993] Zheng, Z. 1993. A Benchmark for Classifier Learning. In *Proceedings of the Sixth Australian Joint Conference on Artificial Intelligence*. World Scientific Publisher (An extended version is available as Technical Report TR474, Basser Department of Computer Science, The University of Sydney, available by anonymous ftp to ftp.cs.su.oz.au in the /pub/tr directory). 281–286.