Prototype Selection for Composite Nearest Neighbor Classifiers

David B. Skalak
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
skalak@cs.umass.edu

CMPSCI Technical Report 95-74

July 1995

PROTOTYPE SELECTION FOR
COMPOSITE NEAREST NEIGHBOR CLASSIFIERS

A Dissertation Proposal 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

July 1995

Department of Computer Science

PROTOTYPE SELECTION FOR
COMPOSITE NEAREST NEIGHBOR CLASSIFIERS

A Dissertation Proposal 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

I thank my committee chair Edwina Rissland for her inspired guidance and inspiring support, and members Paul Utgoff, Andy Barto and Michael Sutherland for their help to me and for the direction that they have given to this research. I appreciate the assistance of David Mix-Barrington, Victor Lesser, Shlomo Zilberstein and Bill Lenhart. Thanks to Oliver Selfridge for his enlightening comments and encouragement. Thanks also to Jeff Clouse and M. Timur Friedman for discussions and to Neil Berkman for having written and provided me with source code for ID3. 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 and in part by National Science Foundation Grant No. EEC-9209623 State/University/Industry Cooperative Research on Intelligent Information Retrieval.

Abstract

PROTOTYPE SELECTION FOR
COMPOSITE NEAREST NEIGHBOR CLASSIFIERS

July 1995

DAVID B. SKALAK, B.S., Union College

M.A., Dartmouth College

J.D., Harvard Law School

M.S., University of Massachusetts Amherst

Directed by: Professor Edwina L. Rissland

This proposal brings together two problems in classification. The first problem is how to design one of the simplest and oldest classifiers, the k-nearest neighbor classifier. The second problem is how to combine classifiers to produce a more effective classifier.

Our immediate objective is to study a classifier that combines the predictions of a set of *complementary* nearest neighbor classifiers using several well-known machine learning algorithms. We use the term complementary to refer to a set of classifiers whose predictions may be combined to yield a classifier with accuracy higher than any of these component classifiers. The resulting architecture is an instantiation of the *stacked generalization* framework discussed by Wolpert [1992]. A central problem of this research is to characterize the senses in which classifiers are complementary and to present algorithms that create sets of complementary nearest neighbor classifiers. We propose algorithms that selectively incorporate different sets of prototypes into nearest neighbor classifiers.

We bias our search for component nearest neighbor classifiers in favor of classifiers that incorporate only small sets of prototypes. In this proposal we provide evidence

that a very small number of prototypes may be sufficient to give good generalization accuracy on several often used data sets. We also show that simple sampling and search algorithms with a stochastic component may be sufficient to find such prototypes.

This proposal gives an introduction to the problems of nearest neighbor classifier construction and combination, reviews related research, gives an overview of the intended framework for stacked nearest neighbor classifiers, and introduces algorithms for nearest neighbor classifier construction and combination. Finally, we propose work to finish the dissertation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The ability to classify is one important facet of intelligence. Given a set of examples that have been assigned a class label, the task of a classification algorithm is to predict correctly the class of unlabeled examples. Many algorithms have been developed to perform this task, which is often called *supervised learning*, but each classification algorithm works well for some data sets but not for others. Given a specific data set, the first problem confronting a user is to select a *model class* for a classifier, such as linear discriminant functions [Nilsson, 1990], decision trees [Quinlan, 1986] or instance-based classifiers [Cover and Hart, 1967].

Once a model class has been selected, *model fitting* must be performed: a specific classifier must be configured for application to the data set. The configuration of a classifier might involve setting parameters (e.g., k, the number of neighbors to be considered in a k-nearest neighbor algorithm) or designing more complex functional sub-procedures (e.g., the rule for combining the predictions of a set of k neighbors). The user is faced conceptually with a large set of classifiers of different configurations that potentially might be applied to the data. The simplest strategy to deal with this surfeit is winner-take-all: define what it means for a classifier to be the best and choose a single classifier that is best in that sense. A typical winner-take-all approach is to pick the algorithm with the highest average cross-validation generalization accuracy.

But picking a single classifier that is somehow the best forecloses the possibility that the losing candidates could contribute to classification predictions. Perhaps another classifier can predict with high accuracy examples of a sort that are poorly classified by the winning classifier, for instance. Perhaps a disagreement in predictions between two strong classifiers suggests the level of confidence that can be associated with either prediction. Perhaps selecting a few classifiers and taking the majority vote of their predictions will be more accurate than any of the sampled classifiers. In order not to foreclose potential advantages such as these, an alternative to winner-take-all is desirable: combine the predictions of a set of classifiers with different configurations. This alternative is the subject of this proposed thesis.

While there are many strategies for combining classifiers, a straightforward approach has been captured by Wolpert in the *stacked generalization* framework [Wolpert, 1992; Wolpert, 1993]. In its most basic form, a layered architecture is created with a set of classifiers forming a first layer and a single combining algorithm forming a second (somewhat degenerate) layer (Figure 1.1). A set of *component classifiers* (called by Wolpert *level-0 classifiers*) in the first layer take an instance as input and each makes a class *prediction*. The component predictions are amalgamated by a *combining classifier* (*level-1 classifier*) and the ultimate prediction of the entire *composite classifier* (*stacked classifier*) is output.

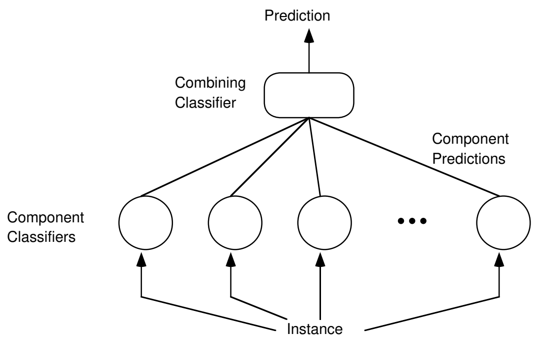

Figure 1.1. Composite classifier architecture

Many open questions are raised by this simple framework[1]. Wolpert has gone so far as to observe:

> 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

[1]We discuss the similarity of this framework to a two-layer neural network in Section 3.7.

> 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. From a broad metaphorical perspective, the goal of the proposed research is to realize some of the promise of the stacked approach to classification by turning black art into science. We can begin to move beyond witchcraft by identifying three subproblems of the general problem of combining classifiers.

Problem 1: Selection of a Combining Classifier. Given a set of component classifiers and a data set as input, output a combining classifier that maximizes the generalization accuracy of the resulting composite classifier.

Problem 2: Search for Component Classifiers. Given a combining classifier and a data set as input, output a set of component classifiers that maximizes the generalization accuracy of the resulting composite classifier.

Problem 3: Integrated Search for Combining and Component Classifiers. Given sets of available combining classifiers and component classifiers and a data set as input, output a combining classifier and component classifiers from those sets that maximize the generalization accuracy of the resulting composite classifier.

Our primary focus in the thesis will be on Problem 2, selecting the component classifiers. Problem 2 is the central, keystone problem of these three. The reason is based in the fundamental observation in artificial intelligence that given the right representation of the data, problem-solving and learning can be easy [Barr *et al.*, 1981]. The role of the component classifiers is analogous to the role of the first layer of units in a neural network: to effect a re-representation of the input data. In the stacked generalization framework each input item is re-represented as a vector of predictions of the component classifiers (the *derived training set*). In keeping with the fundamental observation, we focus on selecting a set of suitable classifiers to make learning from the derived training set easy. Once a suitable derived training set is created, Problem 1 may be solved easily, with predictions combined by an off-the-shelf algorithm. Finally, given an appropriate derived training set, the interaction between the component and combining classifiers — the source of difficulty underlying Problem 3 — may also be lessened.

Insufficient attention has been paid to Problem 2, the selection of component classifiers. The previous researchers who have offered suggestions about solutions have relied primarily on informal characterizations. It has been observed by one researcher that "the biggest gains came when dissimilar sets of predictors were stacked" [Breiman, 1992, p.4] when he stacked different types of linear regressions. Wolpert has suggested, on the basis of a hypothetical example where three component classifiers made exactly the same predictions, that "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" [Wolpert, 1993, p.6]. In Chapter 2 (Related Research) we catalog types of "dissimilar" classifiers that have been combined in previous research. However, the dimensions along which classifiers should be dissimilar in order to serve as effective component classifiers has not been studied closely or characterized precisely. This thesis proposes to begin to fill this gap.

As an initial matter, we suggest a shift in vocabulary, which suggests a shift in perspective that distinguishes previous approaches to component selection from our own. Rather than *dissimilar* component classifiers, we suggest the term *complementary.* Our approach will be to construct the component classifiers with the goal that they be combined. The few previous stacking efforts have identified ways of choosing dissimilar component classifiers, without regard to whether they will actually work well together in a composite classifier. “Dissimilar,” meaning not alike, does not capture adequately this cooperative notion. Dissimilar classifiers can be different in a way that does not aid their combination.

To drive this point home, consider an analogy to a college basketball team. The members of the team are analogous to component classifiers. Team members may be selected according to some criterion of dissimilarity. They may be from different states, they may be of different races, or they may have different majors. None of these differences probably would optimize or even affect their performance together. But if they are chosen to be complementary with the idea that together they should make a winning team, then a good play-maker, a good outside shooter and a good rebounder will be chosen as part of the team. On the other hand, dissimilarity can give rise to complementarity by coincidence: if the players of various heights are selected, for example. But a better approach would rely less on coincidence, by determining the precise sense in which the players' skills should be complementary and then fielding a team according to that determination. So, we argue, should component classifiers be selected.

In addition to complementarity, there are other desired constraints that one might place on component classifiers, and we therefore bias our search for component classifiers in favor of those that satisfy some reasonable related constraints. All other things being equal, one might prefer (a) fewer component classifiers over more, (b) computationally inexpensive ones over more expensive, and (c) simpler component classifiers over more complex ones.

Clearly, one would prefer fewer classifiers, since training and application costs will be lower than for a composite that incorporates more components. Since composite classifiers require the training of a number of component classifiers and a combining classifier, there is a strong desire to limit the number of the component classifiers in a composite system.

Computationally expensive component classifiers that require high training costs or application costs can be a substantial drawdown on resources. For example, a composite classifier that consists of a linear combination of a set of 10 multilayer networks trained with backpropagation (as by Perrone [Perrone, 1993]) can be computationally expensive to implement. Excessive costs also have the detrimental methodological side-effect of slowing the pace of experiments that can be performed.

Finally, we pose a bias in favor of simpler classifiers, which is a reflection of Occam's Razor [Blumer *et al.*, 1987]. A simpler classifier is one that generates a simpler concept hypothesis, which is shorter in some concept representation language [Angluin, 1992]. Where overly complex hypotheses are permitted, the potential for overfitting a training set is present. We propose to mitigate this potential problem by applying simple classifiers. The danger of overfitting is still present with a simple classifier, but ought to be less likely in that simple classifiers admit concept hypotheses that are not complex and admit fewer of them. For some model classes, simple classifiers are less expensive to train and to apply as well.

Putting complementarity together with these three constraints allows us to formulate our proposed contribution. The contribution of this research will be to evaluate the hypothesis:

Effective and efficient composite classifiers can be constructed in many domains using a stacked generalization framework that incorporates a small number of simple component classifiers that are complementary.

The proposed research will provide characterizations of *complementary*, and then provide algorithms that construct classifiers that are in fact complementary according to each characterization.

Thus this research will propose a theory of classifier combination based on component classifier complementarity and to explore the theory's strengths and weaknesses. We propose to investigate the issues involved in this hypothesis by implementing a system that uses heuristic measurements of component classifier complementarity to guide the search for component classifiers.

We have chosen to limit the implementation to three combining algorithms and to k-nearest neighbor component classifiers. The three combining algorithms are voting [Littlestone and Warmuth, 1989], ID3 [Quinlan, 1986], and k-nearest neighbor algorithms [Cover and Hart, 1967]. We justify these choices in more detail later, but previous research has shown that nearest neighbor and ID3 classifiers have different strengths. Voting is a traditional technique for amalgamating categorical preferences in many social and scientific settings.

As a matter of fidelity to the scientific method, we cannot say what our investigations of the hypothesis will yield. If there is insufficient support for the hypothesis, then we shall show where it fails and explain why, to circumscribe the limits of our hypothesis about classifier combination. In view of the dicta from previous researchers about the importance of dissimilarity, then this demonstration would be a contribution in itself. On the other hand, it would be disingenuous not to offer our expected result: that accurate and efficient composite classifiers can be designed using a small number of k-nearest neighbor classifiers.

Indeed, we have preliminary empirical support for the central hypothesis. In Chapter 4 we show that on the data sets tested, classifiers with small numbers of prototypes can outperform classifiers that use all instances as prototypes, resulting in a reduction of on-line storage costs of up to two orders of magnitude. Further, we show that even simple search techniques are adequate to locate sets of prototypes that have good generalization accuracy. We discuss other approaches to prototype selection in Chapter 2. In Chapter 5 we provide preliminary experimental evidence of the accuracy of several algorithms for creating component classifiers.

The problem we propose is at the confluence of two streams of research: nearest neighbor classifier configuration and composite classifier construction. Until now these problems have been treated independently. A methodological contribution of this research will be to demonstrate 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 — selecting prototypes in particular — and exploring the ways to configure nearest neighbor classifiers can offer new classifiers to combine.

1.1 A Guide to the Proposal

In broad outline, this proposal takes a canonical form. The general problem has been introduced. Next, previous research from a variety of related areas is surveyed, followed by a discussion of how previous work can be supplemented. A solution framework is introduced in general terms, which realizes some of the extensions and improvements to previous research. Next, detail for the framework and a partial implementation are presented. Preliminary experiments supporting the utility of the proposed approach are given. Last, work to finish the dissertation is proposed.

The proposal is organized into the following chapters:

Chapter 2: Related Research. We survey the role of prototypes in several disciplines, the selection of prototypes for nearest neighbor classifiers and the combination of classifiers. We describe the framework of stacked generalization for classifier combination [Wolpert, 1992] and the many open problems that it engenders.

Chapter 3: A Classification Framework. We propose a stacked, two-layer classifier architecture of small nearest neighbor classifiers and a combining classifier, introduce several algorithms for selecting component classifiers, and provide motivation by showing that our approach extends previous work and is analogous to successful work in related areas of machine learning.

Chapter 4: Constructing a Nearest Neighbor Classifier. We first consider the problem of creating an independent nearest neighbor classifier that incorporates only a few prototypes. We show that some simple sampling and stochastic search techniques can locate sets of prototypes with good classification accuracy on several data sets. Finally, we show that a measure of clustering of the classes in a data set is correlated with the accuracy of a sampling algorithm to select prototypes.

Chapter 5: Combining Nearest Neighbor Classifiers. Our preliminary algorithms to select complementary nearest neighbor classifiers are described, and we compare several off-the-shelf learning algorithms for classifier combination.

Chapter 6: Proposed Work. In the final chapter, we suggest alternative paths to complete this project.

CHAPTER 2

RELATED RESEARCH

2.1 Chapter Organization

The problems of prototype identification, nearest neighbor algorithm design and composite classifier creation have been considered either directly or by analogy in various disciplines, including cognitive science, case-based reasoning, statistical pattern classification, machine learning and the theory of artificial neural networks. The relation of previous research to this proposal takes several forms. In some cases, related work merely provides part of a broad background of ideas underpinning this research. Cognitive science research into the nature of prototypes is an example. In other cases, related work has a direct bearing on our proposed research, especially where we provide alternative methods to accomplish classical tasks (reducing prototype sets) or use a very general framework that has been previously identified (stacked generalization to create composite classifiers). Other topics are brought into play in a less direct way, often by analogy, as with the analogy of dynamic neural network construction to the incremental addition of component classifiers to a composite classifier.

This chapter is organized to reflect the hierarchy of the objects of study. Treated in order are

1. **prototypes**, which are selected to build ...

2. **nearest neighbor classifiers**, which are combined to build ...

3. **composite classifiers**.

We examine the relevant work on these three topics from the standpoint of several disciplines. We then suggest ways that the previous research can be extended or improved.

2.2 Prototypes

2.2.1 Psychology, Philosophy and Cognitive Science

The examples of a category that are best in some sense are called "prototypes." Prototypes have been advanced as the basis for theories of category representation, category learning, and classification [Smith and Medin, 1981]. The existence and utility of prototypes have been recognized as a response to the inadequacy of the classical model of categorization. The classical model of categorization holds that there is a necessary and sufficient set of criteria that determine whether an object is a member of a category. While this classical model went 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 [Smith and Medin, 1981; Rosch and Mervis, 1975]. One response 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 on the basis of the weighted similarity of the features possessed by the prototype and the example [Smith and Medin, 1981].

Wittgenstein was one of the first to undercut 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 a *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 game example to demonstrate that categories can have extensible boundaries and that categories can have both central and non-central members.

Much of the trail-clearing research in cognitive science 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, production of examples, asymmetry in similarity ratings, asymmetry in generalization, and family resemblances. However, Rosch does not hold that prototype effects mirror a particular category structure based on prototypes. Rather, 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 provide a theory of representation of category structure [Lakoff, 1987]. Lakoff maintains that domain knowledge is organized by humans in structures called *idealized cognitive models (ICMs)* and that prototype effects are side-effects of that organization [Lakoff, 1987].

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, 1978; Rissland, 1981]

2.2.2 Case-Based Reasoning

In general, case-based reasoning (CBR) systems retrieve similar cases for further analysis or adaptation [Rissland, 1989], not usually for classification. However, a number of case-based reasoning systems have relied on prototypes, usually for memory organization. In these systems a prototype is used to index cases that are similar to the prototype. Typically, prototypes have been used in 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 by the prototype. We consider the role played by prototypes in several CBR systems.

McCarty and Sridharan proposed a representation for legal cases as consisting of legal prototypes plus possible deformations of them [McCarty and Sridharan, 1982]. Their approach anticipated, from a legal standpoint, Lakoff's view from cognitive science that some categories can be represented as a radial structure with a prototype at its hub, rules that generate other category members at the spokes, and other category members along the rim [Lakoff, 1987, pp.83-84]. More recently in the law, Sanders applied a notion of example prototypes plus deformations of so-called *safe harbor* plans to creating plans for income tax transactions [Sanders, 1994]. Safe-harbor plans are stereotypical plans to achieve a certain result under the tax code.

The ReMind CBR 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.

PROTOS is an example of a CBR reasoning system that does rely on case prototypes for classification, the diagnosis of ear diseases [Bareiss, 1989]. Prototypes in PROTOS 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 as the cases to which comparison should be made first. We next consider briefly the role of prototypes in machine learning.

2.2.3 Machine Learning

The interest of machine learning researchers in prototypes understandably has been on how they can be used for classification, since classification is a fundamental task studied in machine learning, rather than on their inherent interest. Therefore we postpone a discussion until Section 2.3, where we discuss the selection of prototypes for nearest neighbor classification. However, some machine learning research has exploited the notion that some instances exhibit a high degree of typicality and this research warrants brief mention here.

Prototypes are often thought to be instances with large typicality in a domain. 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'." [Minsky, 1965]. Rosch's family resemblance measure [Rosch and Mervis, 1975] is a primary example of a typicality measure, and it has been used to identify prototypes for classification tasks by Zhang [Zhang, 1992]. This typicality measure is based on the statistical distribution of instances in the class and 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. In other implementations, the reliability of an instance when used for prediction can be implemented as a weight attached to each instance in order to give a graded structure to a category [Aha, 1990; Salzberg, 1991].

2.2.4 Artificial Neural Networks

Two artificial neural network algorithms related to prototype identification are learning vector quantization and radial basis functions. Learning vector quantization is used to locate prototypes; radial basis function networks require the identification of prototypes in order to perform function approximation.

Vector quantization is an 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 [Kohonen *et al.*, 1988; Hertz *et al.*, 1991].

Radial basis function (RBF) networks also provide an analogy for prototype selection [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 points in the input space, often called the *centers* (since they are the centers of the often radially symmetric basis functions). The function values at new points are a weighted sum of the Gaussian values, weighted by the distance of the new point from the centers.

While the details of RBF networks are not relevant 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 placing the centers, including the computationally expensive technique of using all input instances as centers [Wasserman, 1993], simple clustering methods [Burrascano, 1991], LVQ [Kohonen *et al.*, 1988], and applying supervised learning algorithms [Wettschereck and Dietterich, 1992]. Wasserman [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.

In this section we have surveyed briefly the roles that prototypes have played in a variety of disciplines. In the next section we show how prototypes have been used for one particular task, building nearest neighbor classifiers.

2.3 Constructing Nearest Neighbor Classifiers

In this section we deal with research that is related to the intermediate level (Figure 1.1) of our classification hierarchy: nearest neighbor classifiers, which are built from prototypes, and which will be used to construct composite classifiers.

In this proposal, our primary technique for increasing classification accuracy, as well as reducing the computational expense of nearest neighbor computation, is to select a proper subset of the training instances to be used as prototypes. Two other general classes of methods have been applied to the subgoal of reducing the computational expense of nearest neighbor retrieval: (1) improved data structures for neighbor retrieval, particularly *k-d* trees [Moore, 1990], and (2) feature selection [Cardie, 1994; Skalak, 1994; Tan and Schlimmer, 1990]. Improving the data structures used for retrieval will not in itself improve the accuracy of classification, just the speed. Feature selection has been studied for a long period by the statistics, pattern recognition and machine learning communities [Krzanowski, 1988]. Our focus for this subsection is on the characteristics of the algorithms that have been used to find prototypes for nearest neighbor classification.

Reducing the number of prototypes used for nearest neighbor retrieval has been a topic of research in the pattern recognition and machine learning communities for 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, mostly 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. Useful surveys of such editing algorithms have been undertaken by Aha [1990] and Dasarathy [1991].

The term "prototype" has been adopted by a number of different research communities and the result is ambiguity in the application of the term. Whereas in cognitive science the term is used to refer to a distinguished example, in pattern recognition research the term is used to refer more generally to any reference instance used in a nearest neighbor classifier, in particular those that remain after the application of some editing algorithm. The term prototype is ambiguous because the connection between the cognitive science and pattern recognition notions of a prototype has not been precisely characterized. This connection could provide grounds for useful research into whether distinguished examples for a domain are the ones that are best retained as reference instances in classifiers.

In both research communities the goal of editing a set of instances is to reduce the expense of nearest neighbor computation without sacrificing generalization accuracy. Examples of the various approaches to the problem 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]. Other systems deal with reference selection by storing averages or abstractions of instances [Chang, 1974; de la Maza, 1991]. In the next two sections, we survey important representatives of editing algorithms.

2.3.1 Pattern Recognition Editing Algorithms

In this section, we review the major algorithms for creating nearest neighbor classifiers with a reduced set of prototypes designed during the first wave of interest in this topic. As many techniques as there are synonyms for "edited" have been created, including *condensed*, *selective* and *reduced* algorithms. In addition to similarity in their names, many of the algorithms for achieving a reduction have relied on similar procedures.

Classical editing algorithms can be characterized according to whether the algorithm works from the *top-down*, starting with the entire data set and filtering the instances according to some rule to arrive at a set of prototypes; or from the *bottom up*, starting from the null set of prototypes or a random sample of instances, one from each class, and adding prototypes one-by-one according to some rule.

Table 2.1 collects the major editing algorithms developed by researchers generally associated with pattern recognition research. We describe each 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 [Hart, 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.

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.

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.

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 nearest algorithm for the classification of new instances.

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.

Unlimited and All k-NN Edited Nearest Neighbor. Tomek provides two variants of the Edited Nearest Neighbor algorithm (ENN) [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 $i = 1, 2, \ldots, k$.

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 using the instances in the next subset in the partition as prototypes, and applying a 1-nearest neighbor algorithm. 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 have resulted in no editing, the loop is exited and the algorithm outputs the remaining instances.

2.3.2 Machine Learning Editing Algorithms

Since the foundational work done around the 1970's on editing algorithms, researchers from machine learning have re-visited the 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 of course a generalization of the 0-1 weights implicitly applied by algorithms that only retain or delete an instance in the prototype set.

Table 2.2. 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

IB2 and IB3. Aha's IB2 algorithm, which is very similar to the Reduced Nearest Neighbor algorithm, provides one way to reduce storage requirements [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 of Aha, IB3, uses only *acceptable*, non-noisy instances as prototypes. To determine if an instance is acceptable, first an instance is provisionally accepted if it is misclassified according to the (applicable symbolic- or numeric-class) criteria of IB2. Only those acceptable instances that display statistically significant predictive accuracy are ultimately retained and allowed to participate in predictions, however. To determine statistical significance, prediction records are maintained for stored instances that record 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 accurate classification.

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 distance from other 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. Zhang found that BIBL performed quite poorly on some data sets.

All of the above algorithms output a subset of (possibly weighted) instances from the original data set. A different approach is exemplified by the research of Chang [1974], which creates artificial prototypes that are not "real" instances. The idea is to start with every instance in a training set as a prototype, and then successively merge (in some fashion) any two nearest prototypes of the same class so long as the classification accuracy is not downgraded. The PROTO-TO system of de la Maza [1991] also creates artificial prototypes, which de la Maza calls "augmented." 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 included in the prototype sets of instance-averaging algorithms. They found that instance-filtering algorithms were more appealing on that basis.

In this section we have reviewed many techniques for editing a training set to yield a set of prototypical instances for nearest neighbor classifiers. In the next section we move up the hierarchy to survey previous classifier combination research.

2.4 Combining Classifiers

In this subsection we focus on the methods that previous researchers have used to combine the classification predictions of a set of classifiers. The problem of combining classifier systems has received a large amount of recent attention (e.g., [Wolpert, 1993]), but has been a recognized area of research for quite a long time. In seminal artificial intelligence research such as Selfridge's Pandemonium [Selfridge, 1959] and Nilsson's committee machines [Nilsson, 1990](originally published in 1965), the idea of combining classifiers was advanced. In 1989, Clement reviewed over 200 papers on the more general issue of combining forecasts [Clement, 1989]. Particular research interest recently has been shown in the combination of neural classifiers (e.g., [Edelman, 1993; Jacobs *et al.*, 1991; Jordan and Jacobs, 1993; Perrone, 1993].)

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 combined by a *combining classifier*.

Because classifier combination is such a large subject, a fine screen is required to unearth previous work that is relevant to the proposed research, and to discuss those salient aspects in sufficient detail so that the contributions of the proposed efforts will be clear. An example of a line of research that is not particularly germane is the subfield of machine learning that has emerged to deal with the integration of different inferential strategies to solve problems, called *multistrategy learning* [Michalski, 1994]. Strategies encompass broad classes of rational methods in this terminology, such as empirical generalization, constructive induction, deductive generalization and explanation [Michalski, 1994, p.4]. The problem of multistrategy combination is broader than our effort, since we apply only a single inferential strategy, inductive concept learning.

In order to maintain a clear focus in the following survey, we shall try to provide answers to the following questions, where they are applicable:

1. Are the component classifiers assumed to be given *a priori* or is a theory provided as to how to select them?

2. What is the scope of each component classifier? Is it used to classify the entire instance space or a proper subset of it?

These particular questions are relevant because the answer to one or both of them will distinguish our proposed line of research.

Methods for selecting classifiers to be combined and combining their predictions can be grouped into five categories, which reflect the approaches of five research fields:

1. Hybrid systems that integrate case-based and other methods from *case-based reasoning*
2. Stacked generalization and recursive partitioning (divide-and-conquer) methods from *machine learning*
3. Modular networks from *artificial neural network theory*
4. Stacked regression from *statistics*
5. Voting algorithms from *computational learning theory*

We consider, in order, progress on the combination of predictions contributed by these disciplines.

2.4.1 Hybrid Case-Based Reasoning

While the tasks performed by case-based reasoning systems are often not simply classification tasks, the combination of case-based systems and other algorithms has been a fairly active area of research, including 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 module for mechanical design) [Goel, 1989], FRANK (case-based and OPS5 system in planning framework for diagnosis tasks) [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]. The emphasis in these projects was frequently on the control strategies or the implementation framework for integrating the component reasoning modules. In some cases these control regimes are more sophisticated than the straightforward control strategies contemplated by the composite classifiers we shall propose because the hybrid case-based systems apply larger amounts of domain knowledge, incorporate more extensive case representations, or perform more complex tasks.

Case-based systems that do combine 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.

2.4.2 Machine Learning

Two general approaches to combining classification algorithms can be distinguished:

1. **Recursive Partitioning:** Systems and algorithms in which the component classifiers recursively partition the instance space into subspaces. Subspaces that are in the final partition are assigned class predictions for all of the instances in that subspace. Algorithms of this type are also called *divide-and-conquer* algorithms.

2. **Global Scope:** Systems and algorithms in which each individual classifier makes a classification prediction for every instance and 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.

Thus in global scope systems the component classifiers are applied to all the instances in the space, regardless of where they reside. In recursive partitioning algorithms, each classifier is applied only to a local, (usually) proper subset of instances. Recursive partitioning classifiers are generally implemented as decision trees, where a classifier is present at each internal node of the tree, and its task is to learn a subconcept for the instances that fall to that node. Work by Utgoff and Brodley provide paradigm examples of this approach [Utgoff, 1989; Brodley, 1992]. 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 the concept descriptions that they apply [Utgoff, 1989]. In the context of class probability trees (which return a vector of class probabilities at their leaves), Buntine argues that combining multiple trees can be interpreted as an approximate method to compute theoretical values that arise from a Bayesian decision-theoretic model of classification [Buntine, 1991]. Langley has provided a framework for creating a tree of Bayesian classifiers [Langley, 1993]. While recursive partitioning has clear strengths, the research proposed here will focus on systems that have global scope. We discuss the reasons for this design choice in Section 2.5, where some of the disadvantages of recursive partitioning algorithms are discussed. These disadvantages include the availability of fewer training instances in each subspace, the presence of hard boundaries between subspaces and the inability to learn from the mistakes that a classifier may make in a subspace.

The primary framework for global scope classifier combination is stacked generalization, to which we turn next.

2.4.2.1 Stacked Generalization

Much of this proposal is an outgrowth of research on a technique called *stacked generalization*, a framework for combining classifiers [Wolpert, 1992; Wolpert, 1993]. Wolpert concentrates on a simple, 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 of a layered classifier is not original to Wolpert, however, and can be traced at least as far back as 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].

The idea of stacked generalization is quite simple. The algorithm assumes that we have been given a set of level-0 classifiers, a level-1 classifier, and a data set of classified examples with real-valued features. The classes are real values as well under Wolpert's assumed framework. The architecture has the usual two phases, training and classification.

Training Phase: There are three steps.

1. First, apply leave-one-out cross validation as follows. For each instance in the data set, train each of the level-0 classifiers using the remaining instances. After training, classify the instance left out 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.

2. The resulting set of vectors of predictions forms a new data set, which is the training set for the level-1 classifier. Train the level-1 classifier on this derived training set.

3. Re-train the level-0 classifiers on the entire training set.

Classification 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 original instance.

Although it is not guaranteed to increase generalization accuracy, stacked generalization can be an effective meta-learning technique to boost generalization accuracy. In at least one case stacked generalization has attained accuracy at levels not achieved by other learning algorithms. According to Wolpert [Wolpert, 1993, p.7 draft], the current best system on a difficult protein folding prediction task employs stacked generalization [Zhang *et al.*, 1992]. Wolpert has applied the stacked generalization framework to the NetTalk problem of phoneme prediction from windows of letters. The experiment was designed to test "whether stacked generalization can be used to 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 distance-weighted Euclidean distance metric. 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. That is, each classifier had a one-dimensional input space, and differed only in the letter slot that they 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%. However, the accuracy of the stacked generalizer was not compared to other machine learning algorithms[1].

Schaffer has investigated an extension of stacked generalization, *bi-level stacking*, in which the combining classifier also has access to the original feature vector, and not just to the predictions of the combining classifiers [Schaffer, 1994]. The intended benefit is to allow the combining classifier to take the original values into account to determine how to arbitrate among the component classifiers. For example, component classifier 1 might be more reliable when the stock market is trending upwards, while classifier 2 might be more reliable when the market is trending lower. Schaffer performed an experiment to compare stacking with bi-level stacking on five UCI data sets. The three model classes combined were a decision tree, rule induction, and a neural network. The combining classifier was drawn from each one of these classes

[1] 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.".

as well. In only 3 of 15 experiments (using three combining classifiers on each of five data sets) did bilateral stacking outperform stacking. This empirical result raises doubt about the utility of this implementation of bilateral stacking for most of the problems considered by Schaffer in these experiments.

The work we propose will instantiate, complement and extend aspects of Wolpert's stacked generalization framework. We shall return to the topic of the relationship of the proposed research to stacked generalization in Section 2.5. In the next section we continue our survey of related research by examining some techniques from the theory of artificial neural networks that are related to classifier combination.

2.4.3 Artificial Neural Networks

There are two specific strains of neural network research that are related to this proposal:

Neural network combination algorithms. There have been several implementations for combining neural network classifiers, including the hierarchical mixture of experts algorithm [Jordan and Jacobs, 1993; Jacobs *et al.*, 1991], layered radial basis function classifiers [Edelman, 1993] and an array of Restricted Coulomb Energy Network classifiers [Cooper *et al.*, 1982]. While these methods are not usually considered instance-based techniques, the algorithms we will discuss do involve neural architectures that incorporate examples to a greater extent than other neural network algorithms such as backpropagation in a multilayer feedforward network. We also survey non-instance-based research that has focused on classifier combination.

Network construction algorithms. Techniques for changing the architecture of a neural network dynamically are relevant by analogy to methods we will propose for adding classifiers dynamically to a composite classifier.

There are also general analogies between the layered stacked generalization framework and a multilayer feed-forward network, which we discuss further in Section 3.7. By analogy, a hidden unit of a multilayer net corresponds to a component classifier.

This analogy is borne out in several respects: both are computational units, whose output can be interpreted as a category assignment of the input (especially where the output of a hidden unit is squashed, thereby driving the output towards the endpoints of its output range). In addition, where the role of the component classifiers is not merely to output correct predictions, but to output predictions from which a combiner will learn easily, the cooperation between component classifiers is like the cooperation between the units of a neural network.

The role of hidden units is described by Hertz, Krogh and Palmer [1991]: "In some cases the emergence of significant internal representations is also observed; some of the hidden units discover meaningful features in the data (meaningful to humans) and come to signify their presence or absence." [p. 133] No reference we have encountered has described an analogous ability of stacked classifiers to detect emergent features, however.

We discuss these network combination and construction techniques below.

2.4.3.1 Network Combination Techniques

As in the work in machine learning, the combination of neural networks may be characterized according to whether the component networks are designed to classify only a proper subspace (subspace algorithms) of the input space or to classify the entire input set (global scope algorithms).

Subspace Algorithms.[2] The work of Jordan, Jacobs and Barto [Jordan and Jacobs, 1993; Jacobs *et al.*, 1991] on hierarchical mixtures of experts is one strain of the composite algorithms. In their research, 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 to enable task decomposition. Thus the combination algorithm applies a gating network that learns

[2] In contradistinction to global scope algorithms, we have used the term "subspace algorithm" here, rather than the term previously applied, "recursive partitioning." Here the algorithms create a single partition of the space, and do not partition recursively.

which expert network to devote to which input. Nowlan and colleagues have proposed a similar framework [Nowlan, 1990].

Global Scope Algorithms. 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 are unanimity, majority vote, averaging and several mechanisms based on thresholding the confidence of the component networks in their classification prediction. Battiti and Colla note that a "sufficiently large uncorrelation in ... mistakes" [Battiti and Colla, 1994, p.691] leads to boosted classification accuracy. They vary 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 assumption is not justified that the predictions of the resulting networks are independent. While it is not surprising that this assumption is not true, they base their theoretical analysis on this assumption. Their experiments are conducted in the realm of optical character recognition (OCR), where a reject option is typically also available to a system.

Although his approach is not called stacked generalization, Edelman [1993] instantiates a stacked generalization framework in which the predictions of a set of level-0 radial basis function classifiers are combined using a level-1 radial basis function classifier. This composite architecture is applied to the task of face recognition, where 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.

Cooper and colleagues have been granted a patent on a classification method called *NLS* for *Nestor Learning System*[3] [Cooper *et al.*, 1982; Wasserman, 1993]. NLS consists of an array of Restricted Coulomb Energy[4] System (RCE) classifiers

[3] Nestor Learning System is a trademark of Nestor Inc.

[4] Restricted Coulomb Energy is a trademark of Nestor Inc.

whose classification decisions are combined using a so-called Class Selection Device (CSD). An RCE classifier is a descendant of classifiers that are inspired by ideas from electrostatic field theory. 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 the classification phase, the prediction of the highest priority classifier that gives an unambiguous response is used.

The CSD relies on a priority system that uses the highest priority classifier that produces an unambiguous response. (In the NLS algorithm, classifiers have the option to output "confused," similar to a reject option in OCR applications.) Otherwise, if there is no classifier that gives an unambiguous response, "a simple vote counting procedure" is used [Reilly *et al.*, 1987, p.II-501].

2.4.3.2 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 proposal 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 overgeneralization. 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 shall propose for component classifier configuration. 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.

Finally, model selection and combination have also been faced in statistical research. In the next section we consider how those lines of research relate to the work we propose to undertake.

2.4.4 Statistical Approaches

There are two basic approaches to the combination of statistical models *winner-take-all* and *model combination*. In the statistical literature, statistical models are often called *estimators*, *predictors* or *generalizers*.

1. Winner-take-all. Examples of the statistical approach to classifier selection that emphasize the role of the training sets are cross-validation[5] and bootstrapping[6] [Efron, 1979]. These approaches sample the training set to estimate the average generalization accuracy of a classifier. In a setting where multiple classifiers are under consideration, these techniques determine which has the highest estimated generalization accuracy, and simply use that classifier on the data set [Wolpert, 1992]. These techniques are therefore examples of a simple approach to classifier "combination": winner-take-all. For example, the idealized Pandemonium model of Selfridge [1959] was a winner-take-all system in which a *decision demon* selected the *cognitive demon* whose output (or "shout") was the greatest ("loudest") from among those demons, each of which represented a particular pattern [Selfridge, 1959, Figure 1, p. 515].

2. Model combination. Research on combining statistical predictors has concentrated on linear combinations of the predictors [Breiman, 1992; Perrone, 1993]. Breiman investigated the stacked generalization framework using linear combination, and calls his instantiation *stacked regression*. Breiman also investigated the combination of several types of predictors: classification and regression trees (CART trees) and two types of linear regressions: subset and ridge regressions. Diverse CART trees for combination were produced by growing a large tree and then pruning it to include distinct numbers of leaves. Subset regression uses subsets of features in a linear regression function. In the subset regression experiments, Breiman combines predictors gotten by stepwise backward deletion of variables. Ridge regression is a variation that is used for highly correlated variables, as is often the case for a set

[5]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 remaining $n-1$ subsets constitute the training set. Generalization accuracy on the withheld test set is computed for each ply, and the average over the n plies is taken.

[6]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.

of predictors that are designed to approximate the same function. It attempts to minimize the sum of the residual-sum-of-squares plus a parameter, λ, times the sum of squares of the regression coefficients. 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." These results are not described further, however.

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 [Perrone, 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. Perrone used ten multilayer perceptron networks with varying numbers of hidden units and different random initial weights in his experiments on optical character recognition with good results. On a classical time series prediction problem, sunspot prediction, ten multilayer perceptron networks with the same number of hidden units were combined.

Statistical methods have thus focused on winner-take-all methods or the linear combination of generalizers, themselves often linear regressions. While we have treated "statistical methods" and "artificial neural networks (ANNs)" in separate sections, linear combination methods have prevailed in both settings. Of course the outputs of neural nets are combined in research that falls under the "ANN" heading, rather than the outputs of statistical regression algorithms. See Section 2.5.2 for a description of neural network combination research. Making the assumption that the combiner is linear does allow more formal analyses of the algorithms applied in both research settings, however. In the next section, which is the final section on related work on classifier combination, we discuss the contributions of computational learning theory, whose focus is on formal description.

2.4.5 Computational Learning Theory Algorithms

Angluin describes the goal of computational learning theory (COLT) as "Give a rigorous, computationally detailed and plausible account of how learning can be done." [1992, p.351]. She identifies voting schemes as one class of techniques for constructing learning algorithms.

The *Halving* and *Weighted Majority* algorithms of Littlestone and Warmuth [Littlestone and Warmuth, 1989] are the two major algorithms in this area. The theoretical framework and terminology for the COLT approach to the problem 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 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.

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. If there is a consistent algorithm in the pool, the Halving 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 weights are associated with each algorithm in the pool. The WM predicts according to whether the total weight of the algorithms predicting 0 is greater than the sum of the weights of the algorithms predicting 1. The learning 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 an upper bound on the number of mistakes made by the composite algorithm that are $O(log|\mathcal{A}| + m)$ where there exists an algorithm in the pool that makes at most m mistakes.

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 this proposal we emphasize the *closed fusion* approach. In the closed fusion case, Rao and colleagues demonstrate 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).

This concludes our introduction to related research. We next discuss how our proposed work will supplement previous research.

2.5 Limitations of Existing Research

In this section we consider ways that the research proposed here can extend some of the previous efforts we have described in this chapter. This section is divided into two subsections, nearest neighbor editing algorithms and classifier combination. In the next subsection we discuss how the search bias of previous nearest neighbor editing algorithms is (a) inappropriately limited and (b) not well suited to building classifiers that are to be combined. In the second subsection, we observe that many fundamental aspects of Wolpert's stacked generalization are not understood. As we said in Chapter 1, the desire to understand how this very general framework can be instantiated is part of the motivation for this thesis.

2.5.1 Nearest Neighbor Editing Algorithms

In this section we discuss how the search bias of previous nearest neighbor editing algorithms is unduly limited with respect to two goals. (1) The bias limitation does not help achieve the original, stated goal of these algorithms: to select a minimal consistent prototype set (or at least to find the smallest set that achieves a given level of accuracy). Recall that a minimal consistent subset is a smallest set that correctly classifies 100% of the training instances. (2) The bias does not aid our own goal of building composite classifiers from a set of component nearest neighbor classifiers. We treat the two goals in turn.

1. Building an Independent Nearest Neighbor Classifier. None of the existing editing algorithms is guaranteed to find a minimal consistent subset. Therefore, there is still room for fresh research on prototype selection, notwithstanding the attention that has been given to this topic. However, for some datasets, an existing algorithm 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.

Nevertheless, existing editing algorithms that do not rely on instance weighting immediately and only change the *cardinality* of the set of prototypes when some triggering criterion occurs, such as a misclassification. 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. Thus the potential is present for inadequate search for prototype sets of a given, small cardinality. Algorithms with the potential to sample better the space of small sets of prototypes would be useful.

2. Building Component Nearest Neighbor Classifiers. Secondly, existing editing algorithms will not be useful for prototype selection for the simple reason that the objective of the present research is different. The objective of previous research was to build a single minimal classifier of maximal accuracy (with some presumed tradeoff between size and accuracy). Our objective is to construct a set of several small nearest neighbor classifiers in order to use them as components in a composite classifier. There are two reasons why the difference in goals is important.

First, each editing algorithm can output only one set of prototypes, and therefore can be used to build only one classifier. Combining multiple copies of the same classifier into a composite classifier will not increase classification accuracy. There may be *ad hoc* ways to alter existing algorithms to produce multiple sets, but we require algorithms that can produce multiple sets of instances as candidate prototype sets for complementary classifiers.

Second, the objective of previous editing algorithms was to build an independent classifier of maximal accuracy. Since the classification accuracy of a component classifier is not the only criterion we shall apply in selecting classifiers, our approach to composite classifier construction requires a prototype selection method that incorporates other selection criteria into the body of the algorithm to choose what instances to retain. Selection algorithms should also choose prototypes that are appropriate for component classifiers that are going to be combined with others and are complementary. Since previous editing algorithms only apply criteria based on classifications or misclassifications, it is not clear how they can be extended to include other fitness functions without substantial damage to the existing algorithms. In any event probably they would have to be greatly extended.

2.5.2 Combining Classifiers

In this section we discuss previous work on classifier combination, concentrating on the many open questions engendered by the stacked generalization framework. But first we note several reasons that we have not adopted one tried-and-true approach to classifier combination, recursive partitioning.

There are a number of advantages to recursive partitioning algorithms, in particular the ability to define a classifier that is locally appropriate for particular regions of the instance space [Utgoff, 1989; Brodley, 1992]. On the other hand, there may be disadvantages to recursive partitioning. One disadvantage is that fewer instances are available to train a classifier, since each is applicable only to a proper subspace of the instance space. This training set limitation is an important consideration in domains where only a small amount of data is available. Fewer training instances may increase the likelihood of overfitting a small region, especially where models with many degrees of freedom are applied. Secondly, most recursive partitioning algorithms have hard boundaries between subspaces, which may artificially divide the space, making distinctions between instances that are quite close in the instance space. Third, since only one classifier is applied to each subspace, the possibility is foreclosed that other classifiers may provide useful classification predictions about instances that fall into that subspace. For these reasons we have adopted a combination method, stacked generalization, that permits each classifier to be trained with all available data and permits each classifier to make a contribution to the class prediction for every instance.

The research by Wolpert on stacked generalization provides the primary framework for the proposed research. We noted in Chapter 1 that stacking is still magic. Important issues still remain to be treated, particularly our Problem 2, the selection of component classifiers. In our survey of previous work we have shown that previous researchers implicitly have assumed that the classifiers that are to be combined have already been given and the problem is how to combine them, or that the component classifiers have been chosen for diversity, which may or may not make the chosen classifiers complementary. In the only reported work on stacking component nearest neighbor classifiers, Wolpert's NETtalk experiment, the components were distinguished only because they received different inputs. Table 2.3 summarizes the aspects of classifiers that have been varied to create dissimilar component classifiers for stacked generalization.

Table 2.3. Component classifier selection methods

Researcher	Component Algorithm	Aspect Varied
Breiman	Subset regression	regression variables: backward stepwise deletion
Breiman	Ridge regression	ridge parameter λ
Breiman	Subset, ridge regression	λ and regression variables
Breiman	CART trees	pruned to different numbers of nodes
Wolpert	HERBIE	input features
Wolpert	unspecified	partitions of training set
Perrone	Multilayer perceptron	random weight initializations
Battiti *et al.*	MLP, LVQ	random weight initializations, architecture

Several researchers have commented on the desirability of combining dissimilar classifiers. Breiman observed that the "biggest gains came when dissimilar sets of predictors were stacked" [Breiman, 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" [Wolpert, 1993, p.6]. As suggested by Table 2.3, general attempts have been made to create diverse classifiers without the specific intent to create complementary classifiers. We begin to rectify this situation in the remaining chapters.

On the issue of Problem 1, the selection of a combining classifier, the default combining method for this problem and other categorical problems is the plurality vote. Voting is often justified because it is simple, seems fair and, depending on the reader, appeals to democratic principles. However, voting is subject to hidden paradoxes. Placing a set of reasonable axiomatic constraints on the voting procedure can yield contradictions, as shown by Arrow's General Possibility Theorem [Arrow, 1963]. While the theorem is stated and proved formally, Arrow 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 [collective] 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].

Voting has other shortcomings as well. In the context of classifier combination, a voting combining classifier is unable to learn from certain mistakes of the component classifiers. If every time classifier 1 predicts "−" and classifier 2 predicts "−", the correct answer is "+", voting will always predict "−" and be wrong. Thus in this research we will explore whether other learning algorithms perform better than voting to combine component classifier predictions for the problems we examine. In Table 2.4 we collect for reference the combining algorithms that have been applied in the stacked classifiers we have surveyed.

Table 2.4. Combining classifier algorithms

Researcher	**Combining Algorithm**
Breiman	linear combination
Wolpert	HERBIE
Perrone	linear combination
Battiti	unanimity, majority vote, linear combination

On the issue of Problem 3, the integrated search for component classifiers and combining classifiers, we have found no reported analysis or progress so far in the stacked generalization framework for symbolic prediction problems. (For function approximation problems, progress on the integrated problem has been made using the hierarchical mixtures of experts framework, which provides simultaneous training of component and gating networks [Jordan and Jacobs, 1993; Jacobs *et al.*, 1991].)

2.6 Conclusion

In this chapter we have surveyed previous work on the role of prototypes for classification, editing algorithms for prototype selection, and the combination of classifiers. The limitations on the application of existing editing algorithms for building component nearest neighbor classifiers were discussed. There is much more work to be done to realize the potential of the stacked generalization framework. In the next chapter we outline one promising approach to instantiating the framework.

Chapter 3

A Classification Framework

3.1 Chapter Organization

We have a number of topics to address in this chapter. Its broad objective is to introduce the approach we propose to constructing composite classifiers, leaving the details to later chapters and, in some cases, to later work. The first cluster of tasks in this chapter is to examine the classes of algorithms we will use to build the component classifiers and to show how we will extend previous work on editing algorithms and stacked generalization. This description builds the stage set for the next two chapters, which provide detailed algorithms and preliminary empirical evidence for our approaches to prototype selection and classifier combination. A second task is to give an example of the instantiated architecture. With an example in place to make the approach clear, our third task is to provide supporting intuition and motivation for this solution approach. We try to provide motivation by showing how our approach is a generalization of a single k-nearest neighbor classifier, how it is analogous to a multilayer neural network, and how it is analogous to other research in machine learning that shows that simple classifiers can be — perhaps surprisingly — accurate.

3.2 Introduction

The general problem we have posed is to design efficient classifiers that demonstrate good classification accuracy. The solution offered by this thesis proposal is to combine a small number of complementary, small nearest neighbor classifiers. Traditional solutions to the model selection problem have looked for "the model class that yields the simplest classifier with the highest degree of accuracy." [Brodley, 1992, p.22]. The direction we take is different, however. The hypothesis we want to examine is that combining potentially less accurate classifiers that are complementary can lead to high classification accuracy. Many mediocre-to-good classifiers are available as building blocks, and they may be easy to find. Rather than search for the most accurate building block classifiers, we search for the sets of classifiers that are complementary. Once complementary classifiers can be found, their decisions can be combined using some fairly simple techniques. Thus while the goal is in part the same as previous work — to build accurate classifiers — the means we propose is not to follow that criterion solely in the construction of the component classifiers.

Efficiency of this classifier stems from the small computational overhead of the component classifiers, which are nearest neighbor classifiers that rely on only a few prototypes. The combining algorithms examined in this proposal are also quite simple: a decision tree algorithm, a nearest neighbor algorithm, and a voting algorithm.

The research presented in this proposal extends previous work on prototype selection by showing that in some domains decreasing the number of prototypes can be pushed quite far indeed — that only several well-selected prototypes can give good classification accuracy. We extend previous work by demonstrating that algorithms that rely primarily on random techniques can perform the task of choosing a small number of salient prototypes. Further we show when a prototype sampling technique works: when the data to be classified are highly clustered by class.

3.3 Composite Classifier Architecture and Algorithm

The architecture of the type of composite classifier we shall investigate is as discussed in Chapter 1. As pictured in Figure 3.1, the classifier has two layers. A set of component classifiers forms the first layer. A single combining classifier forms the second layer. The architecture is straightforward and follows the general framework of two-level stacked generalization [Wolpert, 1992; Wolpert, 1993]. The learning algorithms we present have the usual two phases: (1) training and (2) classification (or "application").

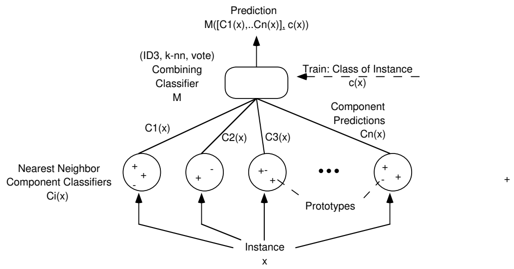

Figure 3.1. Composite nearest neighbor classifier architecture. In the training phase the instance class is supplied to the combining classifier as an additional input.

Pseudocode for the training and classification algorithms is given in Figure 3.2 and Figure 3.3. Our training procedure is basically the same as for stacked generalization, described in Chapter 2. Training instances are input to the component classifiers, whose predictions are collected into vectors of class labels. This set of prediction vectors together with the original class labels forms the training set for the combining classifier. To classify a new instance, first classify it using each component classifier, and then form a vector of predicted class labels. Finally, apply the combining classifier to the vector of predictions.

In the next section we describe algorithms for constructing component classifiers.

3.4 Component Classifier Construction

This proposal introduces four algorithms embodying four approaches to the component classifier selection problem, Problem 2 of the Introduction:

1. **Random Selection:** Sample classifiers

2. **Best Random Selection:** Select the most accurate of a sample of classifiers

3. **Error Orthogonality:** Minimize the errors made in common by classifiers

4. **Inconsistency Reduction:** Reduce inconsistencies in a derived training set.

Key:
T_0: Training set in original representation
T_1: Training set in derived 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

train-composite (T_0, C)

loop for $x \in T_0$
 collect $[C_0(x), C_1(x), \ldots, C_n(x), c(x)]$ into T_1
train(T_1, M)

Figure 3.2. Pseudocode for the training algorithm for a composite classifier.

apply-composite (C, x)

$\mathbf{x} := [C_0(x), C_1(x), \ldots, C_n(x)]$
$M(\mathbf{x})$

Figure 3.3. Pseudocode for the classification algorithm for a composite classifier.

Chapter 5 discusses each algorithm in detail. We describe each briefly here and discuss the reason for including each algorithm in this study. Each of the algorithms is a provisional instantiation of a general approach. In Chapter 6 we propose to refine, speed up and possibly expand this collection.

Random Selection. The pure *Random Selection* algorithm randomly samples sets of prototypes for each component classifier from the training set. The constraint is imposed that at least one prototype be selected from each class, and the desired number of component classifiers (n) is fixed in advance. This algorithm was applied in part because its results may be counterintuitive: that randomly selected classifiers may be complementary. While the Random Selection algorithm is in part included as a baseline, we show in Chapter 5 that this algorithm can return component classifiers whose combination can yield better generalization accuracy than a nearest neighbor algorithm that applies all instances as prototypes. We do not argue that random classifiers are the best component classifiers. However, the performance of stacked architectures that incorporate only a few such random classifiers may be surprising in view of the small percentage of the training set used for classification, the small number of component classifiers and the simplicity of random sampling. These results provide the impetus for further research into the structure of the space of nearest neighbor classifiers.

Best Random Selection. A variant of Random Selection is the *Best Random Selection* algorithm, which incorporates a bias toward prototypes that perform accurately on the training set. This algorithm is included to test the intuition that the search for component classifiers should be biased in favor of components that are more accurate on the available training data. The Best Random Selection algorithm assumes that the most accurate classifiers are complementary. Best Random Selection takes m samples of k prototypes from the training set (sampling at least one prototype from each class exposed in the training set) and returns the n resulting classifiers with the highest individual classification accuracies on the training set.

Error Orthogonality. The *Error Orthogonality* algorithm is based on the notion that classifiers that make different errors are complementary ([Wolpert, 1993; Breiman, 1992; Perrone, 1993]). If the errors made by component classifiers are the same, then a combining classifier will be unable to improve on the component predictions. We discuss the use of the term "orthogonal" in Chapter 5. This algorithm is included to test the intuition that classifiers that make disjoint errors will perform well in a composite architecture, and to test the informal suggestions of previous researchers to stack "dissimilar" [Breiman, 1992, p.4] classifiers, or classifiers "whose guesses are not synchronized" [Wolpert, 1992, p.6]. The current implementation generates a pool of classifiers and then filters them to return classifiers all of whose error sets have a small intersection.

Inconsistency Reduction. The *Inconsistency Reduction* algorithm attempts to reduce a straightforward measure of the inconsistency in the re-representation of the instances that is effected by regarding the instances as vectors of component classifier predictions. The inconsistency of the representation is determined by the extent to which instances from different classes are mapped by the component classifiers to the same derived training instance. This algorithm is iterative and constructive, adding in each iteration a classifier designed to reduce inconsistency ultimately below a user-supplied threshold.

A related algorithm is Saxena's ACR algorithm, which can be applied to select a single classifier on the basis of the encoding of a data set effected by a learning algorithm [Saxena, 1991]. ACR selects a representation based on the Minimum Description Length Principle, which holds that the best hypothesis to describe a data set minimizes the length of the code needed to represent the data. The algorithm that induces the best generalization is the one that results in the shortest representation.

3.5 Generating and Testing Classifiers

Since few systems have relied on sampling a *classifier space* in order to construct a classifier, a few comments may be useful. In our implementation, we sample k-nearest neighbor classifier space by sampling from the training set the prototypes used by a classifier. We assume — for the time being — that the other aspects of the nearest neighbor classifier are fixed, such as the distance metric and the number of nearest neighbors considered, k. In Chapter 6 we suggest expanding search of the space of nearest neighbor classifiers to aspects other than the prototype set.

Three of the provisional algorithms described below use generate-and-test (Random Selection, Best Random Selection and Error Orthogonality) to create an explicit pool of classifiers with different sets of prototypes. These algorithms differ in the way they select classifiers from this pool to be included as components in a composite system. Generate-and-test is a venerable paradigm [Barr *et al.*, 1981]. Clearly this approach is not adequate for many large search spaces. However, where there are many states that satisfy the test criteria, or where many states are equivalent according to the test criteria, generate-and-test may work.

To take a small example, the collection of permutations on a set of size n is $n!$. If, however, an algorithm requires only an even permutation[1] for some purpose (such as an initial seed), generate-and-test may be a viable approach. If many states are equivalent by the test and there are few equivalence classes, generate-and-test may be able to select representatives of the equivalence classes that are implicitly induced by the test criteria. The structure of the space of classifiers from any model class is not well enough understood to know in advance whether classifiers may be successfully generated and tested for individual accuracy (as in Chapter 4) or for complementarity (Chapter 5). Indeed, the question of the structure of nearest neighbor classifier space, and whether it lends itself to generate-and-test, is raised by our positive preliminary results.

Generate-and-test is potentially exhaustive: the classifiers that can be generated, trained and tested using reasonable computational resources may not satisfy the test criteria sufficiently well to be useful. Classifier sampling is made more practical here by constraints on the size and number of component classifiers that have arisen in part from the other considerations discussed in Chapter 1, such as the avoidance of overfitting.

Thus one potential contribution of this line of research occurs as a side-effect of the constraints we have imposed. Small nearest neighbor classifiers are useful building blocks because a simple, known classical approach can be invoked to construct them — generating (by sampling) and testing. We do not argue, incidentally, that small nearest neighbor classifiers are the *only* classifiers that lend themselves to this straightforward means of construction. One segment of a (post-thesis) line

[1] An even permutation on a set can be written as a product of an even number of transpositions of two elements in the set.

of research might be the stacking of shallow decision trees, especially those that incorporate complementary attribute-selection or tree-selection metrics [Utgoff, 1995] or stochastic node splitting criteria.

Having introduced the four algorithms and supplied some justification for the use of generate-and-test by three of the algorithms, we give an example of stacked generalization and then supply additional motivation for the utility of stacking small, complementary nearest neighbor classifiers.

3.6 An Example

In this section we give a simple example to serve two purposes: (1) to show generally how the mechanics of the stacked framework work, and (2) to show specifically that combining classifiers that perform less well on a training set but are complementary can yield learning and generalization accuracy higher than combining more accurate ones that are not complementary.

Suppose that we want to create a composite classifier that combines two nearest neighbor component classifiers on a problem that has three class labels $\{+, -, 0\}$ and data instances that are ordered pairs of real numbers in the real plane. Each of the two component classifiers will take one prototype from each of these classes. Suppose that there are ten training instances (designated by their indexes $\{1, 2, \ldots, 10\}$) and four test instances ($\{11, \ldots, 14\}$). So as not to complicate the example with specific distance calculations, suppose the instances are arranged as depicted in Figure 3.4. It is clear that specific coordinates in the real plane can be assigned to satisfy the constraints that there be three clusters of four instances that are clustered by class ($A = \{1, 2, 3, 11\}$ of class $+$, $B = \{4, 5, 6, 12\}$ of class $-$, $C = \{7, 8, 9, 13\}$ of class 0) and one cluster of two instances ($D = \{10, 14\}$ of class $+$). Further, we assume that the instances in cluster D are nearer to instances in B than to any instance in A, and that the instances in cluster C are closer to instances in A than the instances in D, in terms of Euclidean distance, which we take as the distance metric used by the component classifiers.

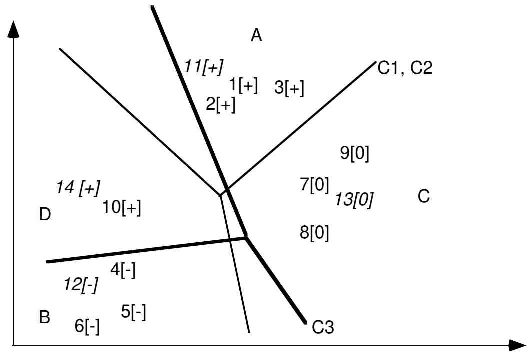

Figure 3.4. Location of data instances in the example. Test instances given in italics. Class labels given in brackets. Voronoi tessellations are shown for the classifiers C_1, C_2 and C_3. In order to keep the diagram clear, the tessellations for C_1 and C_2 are shown as coincident; they are not, but all the instances pictured are partitioned into the same regions by the tessellations for C_1 and C_2.

Consider three 1-nearest neighbor classifiers composed of the following prototypes:

$C_1 : \{1, 4, 7\}$

$C_2 : \{2, 5, 8\}$ and

$C_3 : \{6, 9, 10\}$.

The predictions of classifiers C_1, C_2 and C_3 on the training set are given in Table 3.1.

Table 3.1. Training set classifier predictions.

Instance	C_1	C_2	C_3	**Actual**
1	+	+	0	+
2	+	+	0	+
3	+	+	0	+
4	−	−	−	−
5	−	−	−	−
6	−	−	−	−
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	−	−	+	+

From Table 3.1, it can be seen that the training set accuracy of C_1 and of C_2 is 90% (instance 10 is misclassified by both classifiers); the accuracy of C_3 is 70% (instances 1, 2 and 3 are misclassified).

Now let us build six stacked classifiers, denoted $\mathcal{C}_{12}$-NN, $\mathcal{C}_{12}$-VOTE, $\mathcal{C}_{12}$-ID3, $\mathcal{C}_{23}$-NN, $\mathcal{C}_{23}$-VOTE and $\mathcal{C}_{23}$-ID3. The subscripts indicate the indexes of the component classifiers: $\mathcal{C}_{12}$-NN is a stacked classifier that combines the predictions of component classifiers C_1 and C_2, for example. The suffix indicates the *combining* algorithm, 1-nearest neighbor ("NN"), voting ("VOTE"), or ID3. So $\mathcal{C}_{12}$-NN is the stacked classifier that combines the predictions of C_1 and C_2 using nearest neighbor as the combining algorithm. We will use a calligraphic typeface (e.g., $\mathcal{C}_{12}$) to denote a stacked classifier and upper case italics to denote component classifiers (e.g., C_1).

In Chapter 5 we describe the training and classification procedures for each of these three combining algorithms, but the procedures are straightforward applications of these standard algorithms to the derived data sets. In brief, each algorithm is configured and applied as follows:

Nearest neighbor. The nearest neighbor combining algorithm uses a similarity function that counts the number of exact matches among component classifier predictions. This function is a Hamming distance metric applied to the vectors of class predictions. A vote of all nearest neighbors is taken to yield a final prediction. Typically in the data sets used so far, the nearest neighbors of a test instance at this derived training level match the derived test instance exactly. This means of course that the class predictions of the component classifiers for a test instance are exactly the same as for a previously stored training instance.

Vote. There is no training needed for the voting combining alorithm. A plurality vote of the class predictions of the component classifiers is taken, with ties resolved randomly.

ID3. The version of the ID3 decision tree algorithm uses the information gain ratio test to select atttributes on which to branch, as described in [Quinlan, 1986] and Appendix B. A decision tree is formed from the derived training set vectors, where the tree branches on the predictions of a particular component classifier and the branches correspond to class predictions of that classifier.

The derived training set for each composite classifier consists of a vector of the predictions of each of its component classifiers concatenated with the actual target prediction for each instance. (Where the combining algorithm relies on majority vote, the elements in the derived training set may be thought of as "vote vectors," vectors of the class votes of the component classifiers.) Thus the derived training set for $\mathcal{C}_{12}$ consists of the ten vectors whose components are taken (across rows) from the columns labeled C_1, C_2, and Actual. The first vector component of each derived vector is the prediction of C_1, the second vector component is the prediction of C_2, and the third vector component is the actual target class. The derived representations for the ten training instances for the composite classifiers $\mathcal{C}_{12}$ and $\mathcal{C}_{23}$ are given in Table 3.2. For example, the derived training set for $\mathcal{C}_{12}$ is all the instances in the column labeled "Derived Rep. $\mathcal{C}_{12}$": $\{< +++ >, < +++ >, < +++ >, < --- >, \ldots, < --+ >\}$. Similarly, the derived training set for $\mathcal{C}_{23}$ is formed across the rows of the columns labeled C_2, C_3 and Actual in Table 3.1 and is shown in Table 3.2.

Table 3.2. Derived representations for training instances for composite classifiers $\mathcal{C}_{12}$ and $\mathcal{C}_{23}$

Instance	**Derived Rep. $\mathcal{C}_{12}$**	**Derived Rep. $\mathcal{C}_{23}$**
1	$< + + + >$	$< + \ 0 \ + >$
2	$< + + + >$	$< + \ 0 \ + >$
3	$< + + + >$	$< + \ 0 \ + >$
4	$< - - - >$	$< - - - >$
5	$< - - - >$	$< - - - >$
6	$< - - - >$	$< - - - >$
7	$< 0 \ 0 \ 0 >$	$< 0 \ 0 \ 0 >$
8	$< 0 \ 0 \ 0 >$	$< 0 \ 0 \ 0 >$
9	$< 0 \ 0 \ 0 >$	$< 0 \ 0 \ 0 >$
10	$< - - + >$	$< - + + >$

The decision trees produced by the ID3 algorithm on the derived training sets for $\mathcal{C}_{12}$ and $\mathcal{C}_{23}$ are given in Figure 3.5.

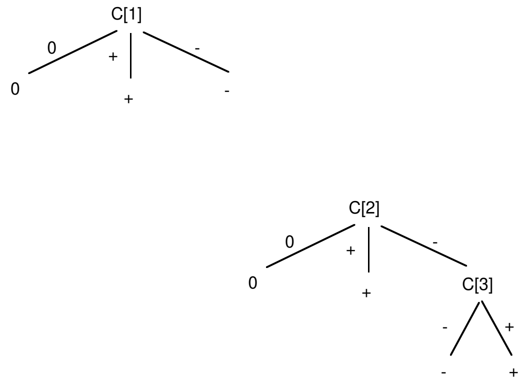

Figure 3.5. ID3 trees for the derived training set for composite classifiers $\mathcal{C}_{12}$ (left) and $\mathcal{C}_{23}$ (right).

The predictions of each of the algorithms are presented in Table 3.3.

Table 3.3. Training set predictions of the six composite classifiers.

Instance	**Actual**	$\mathcal{C}_{12}$-NN	$\mathcal{C}_{12}$-VOTE	$\mathcal{C}_{12}$-ID3	$\mathcal{C}_{23}$-NN	$\mathcal{C}_{23}$-VOTE	$\mathcal{C}_{23}$-ID3
1	+	+	+	+	+	+,0?	+
2	+	+	+	+	+	+,0?	+
3	+	+	+	+	+	+,0?	+
4	−	−	−	−	−	−	−
5	−	−	−	−	−	−	−
6	−	−	−	−	−	−	−
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	+	−	−	−	+	−,+?	+

Now note that composite classifiers $\mathcal{C}_{12}$-NN, $\mathcal{C}_{12}$-VOTE and $\mathcal{C}_{12}$-ID3 only achieve 90% accuracy on the training set, since each misclassifies instance 10.

$\mathcal{C}_{12}$-NN misclassifies instance 10 because the nearest neighbors to 10 $(< - - + >)$ are 4 $(< - - - >)$, 5 $(< - - - >)$, 6 $(< - - - >)$ and 10, all of which match the features of instance 10 $(< - - >)$ exactly. Since there are ties among the nearest neighbors, we take a vote among the neighbors: three votes for class $-$ and one vote for class $+$. Thus $\mathcal{C}_{12}$-NN predicts $-$ for instance 10.

$\mathcal{C}_{12}$-VOTE misclassifies instance 10 because the (unanimous) vote of the component predictions for instance 10 is $-$, since both C_1 and C_2 predict $-$.

$\mathcal{C}_{12}$-ID3 misclassifies instance 10 as well, as shown by the decision tree pictured on the left in Figure 3.5.

$\mathcal{C}_{23}$-NN can be trained to predict the training set with 100% accuracy, on the other hand. The only problematic patterns are instances 1,2 and 3 ($< + \ 0 + >$) and instance 10 ($< - + + >$). But the nearest neighbors to instances 1, 2 and 3 are just those instances, and so the prediction will be unambiguously $+$, which is correct. The nearest neighbor to instance 10 is only instance 10, and so that prediction will be correct as well.

$\mathcal{C}_{23}$-VOTE is subject to the difficulty that where the two component classifiers disagree, the ultimate prediction must depend on the policy for resolving tied votes. There are tied votes for instances 1, 2, 3 and 10. If we assume that the ties are broken randomly, then $\mathcal{C}_{23}$-VOTE will get two of these four instances wrong on average, and score on average 80% on the training set.

$\mathcal{C}_{23}$-ID3 can be trained to predict the training set with 100% accuracy, as shown in the tree depicted on the right in Figure 3.5.

The same reasoning as above shows that $\mathcal{C}_{12}$-NN, $\mathcal{C}_{12}$-VOTE and $\mathcal{C}_{12}$-ID3 achieve 75% accuracy on the *test* set $\{11, 12, 13, 14\}$, since they misclassify instance 14 (and get the other three test instances correct). See Table 3.4. However, $\mathcal{C}_{23}$-NN and $\mathcal{C}_{23}$-ID3, which incorporate a classifier that is only 70% accurate on the training set, show 100% accuracy on the test set. The accuracy of $\mathcal{C}_{23}$-VOTE will depend again on how ties are resolved and may get from 50% to 100% accuracy on the test set, depending on the policy regarding tied predictions.

The derived test sets for the two classifiers are given in Table 3.5.

This simple example demonstrates the important point that a component classifier (C_3) with lower observed accuracy on a training set may be more complementary to

Table 3.4. Test set classifier performance

Instance	C_1	C_2	C_3	**Actual**
11	$+$	$+$	0	$+$
12	$-$	$-$	$-$	$-$
13	0	0	0	0
14	$-$	$-$	$+$	$+$

Table 3.5. Derived representations for test instances for composite classifiers $\mathcal{C}_{12}$ and $\mathcal{C}_{23}$

Instance	**Derived Rep.** $\mathcal{C}_{12}$	**Derived Rep.** $\mathcal{C}_{23}$
11	$< + + + >$	$< + \ 0 \ + >$
12	$< - - - >$	$< - - - >$
13	$< 0 \ 0 \ 0 >$	$< 0 \ 0 \ 0 >$
14	$< - - + >$	$< - + + >$

a given classifier (C_1) than one with higher accuracy (C_2). This observation suggests that the search for component classifiers should be aimed at optimizing criteria other than just training set accuracy. This example further suggests two possible explanations for the superior performance of $\mathcal{C}_{23}$-NN and $\mathcal{C}_{23}$-ID3: (1) C_1 and C_2 make the same errors (although they incorporate disjoint prototype sets), and (2) the problems for $\mathcal{C}_{12}$-NN and $\mathcal{C}_{12}$-ID3 are caused by the presence of inconsistent instances $< - - - >$ and $< - - + >$ in the training set. These two possible explanations respectively give rise to two algorithms we propose, the Error Orthogonality and Inconsistency Reduction algorithms.

In the next section we try to supply additional motivation for considering a stacked classifier that combines the predictions of small, complementary k-nearest neighbor classifiers.

3.7 Discussion and Motivation

The goal of this section is to provide further support for the utility of research into the combination of small nearest neighbor classifiers in a stacked framework. In general, subjective motivation for an approach can be provided in a variety of ways. Providing a visual interpretation of a method, demonstrating that it generalizes an old framework in new ways, making analogies to techniques in allied fields, and formalizing aspects of the approach all can help to enlist support. In the next few pages, we briefly invoke each of these rhetorical methods to show that the proposed solution path is worthy of further study.

Tessellating an instance space. The function of a nearest neighbor classifier is to tessellate an instance space into regions whose boundaries approximate the class boundaries of the instances, a *Voronoi* tessellation. The classical approach to creating such classifiers is to put the computational effort into configuring a single classifier to create the right set of decision boundaries. But a different approach, the one that we advocate here, is to create a set of probably erring, approximate tessellations and to combine the predictions derived from the tessellations. For example, we show that on two of three tested data sets even a random set of tessellations — one created using a small set of random prototypes — is sufficient to yield classification accuracy that is statistically indistinguishable from or better than a nearest neighbor algorithm that uses all instances as prototypes. Choosing the tessellations deliberately to make accurate combination easy provides an alternative to the classical approach of choosing a single tessellation of maximum accuracy.

Generalization of a classical k-nearest neighbor classifier. One perspective on this research is as a generalization of the traditional k-nearest neighbor rule. In one way the generalization is trivial. One nearest neighbor classifier used as a component classifier and the identity function (on class labels) embodied in a combining algorithm is a trivial stacked classifier.

A deeper connection can be made, as well. Suppose that we have a set (of cardinality k) of component k-nearest neighbor classifiers, all of which use the same set of prototypes. Suppose further that each component classifier uses a different rule internally to extract a prediction from the predictions of the nearest neighbors: the ith classifier outputs the prediction of the ith nearest neighbor. See Figure 3.6.

Then, if we use the majority rule as the function to combine the predictions of the k component classifiers, we have the classical k-nearest neighbor algorithm based on the set of prototypes shared by all the classifiers in common. Our proposed framework expands the classical algorithm in at least two ways: (1) it provides that different combining functions may combine the predictions of the k nearest neighbors, and (2) it provides for the component classifiers to apply different sets of prototypes. Thus, the current framework is actually a generalization of the classical k-nearest neighbor algorithm. This interpretation helps to show why a k-nearest neighbor classifier works — because the neighboring instances bring complementary information to bear on the class prediction for a test instance.

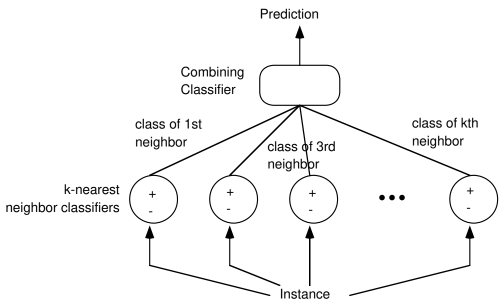

Figure 3.6. Stacked nearest neighbor classifier with k component k-nearest neighbor classifiers as a generalization of traditional k-nearest neighbor classifier

We performed an experiment to test the hypothesis that for some commonly used data sets the prediction of the nearest neighbor may be less accurate on out-of-sample data than the ith nearest neighbor. A 10-fold cross validation was performed on three data sets in which the accuracy of the predictions of the ith closest instance were compared for $i = 1, 2, \ldots, 5$. The percent of instances correct on average across the 10 folds for the ith nearest neighbor classifier is given in Table 3.6.

Table 3.6 shows that for well-known Iris data set, the 2nd, 3rd, 4th and 5th nearest neighbor rules all outperform the 1st nearest neighbor rule. For the Cleveland Heart Disease data, the 2nd, 3rd and 5th nearest neighbors outperform the nearest neighbor. On the Breast Cancer data, the 1st nearest neighbor is indeed the most accurate of

Table 3.6. Percent of test instances correct for nearest neighbor classifiers that predict the class of the ith nearest neighbor, for $i = 1, 2, \ldots, 5$. The symbol "†" denotes statistically significant improvement over the baseline 1-nearest neighbor algorithm (1-NN) at the 0.1 confidence level; "*", significance at the 0.05 confidence level. A two-sample t-test for statistical significance assuming equal population variances is used.

Data Set	**1st-nn**	**2nd-nn**	**3rd-nn**	**4th-nn**	**5th-nn**
Iris	91.3	96.7†	94.0	92.0	92.0
Breast Cancer	68.9	64.3	67.5	67.1	64.3
Heart Disease	76.0	80.0†	81.0*	74.3	77.3

these rules. While a test on so few data sets is not conclusive, these results do provide a basis for additional research into different rules for combining the predictions of a set of neighboring instances.

Networks of nearest neighbor computational units. The stacked generalization framework diagram looks like a multilayer neural network diagram. 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[2]. 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 the proposed 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.

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

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 4, but can also be combined into layered networks (Figure 1.1), 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.

The utility of simple classifiers. An analogy can be made to Holte's research demonstrating that a very simple classification rule is sufficient to perform quite good classification on a number of commonly used databases [Holte, 1993]. Holte showed that one-layer decision trees that classify on the basis of a single attribute can be surprisingly accurate on the tasks drawn from the U.C.I. Machine Learning Repository [Murphy and Aha, 1994]. Our results complement that work by showing how small prototype sets work well on several datasets. While Holte used a decision tree for concept representation and applied several simple inductive learning algorithms, instead we use sets of prototypical instances to provide concept descriptions in a simple nearest neighbor classification algorithm.

One strain of objection that can be made against Holte's results is that the shallow decision tree (so-called *decision-stump*) classifiers he develops have good, but often not excellent, classification performance. Since the primary goal is to create accurate classifiers, this objection goes, Holte's results are interesting but not useful. By analogy, on its face this objection can be made against our own approach to building small nearest neighbor classifiers, which have good but usually not the best generalization capability. Our response to this objection is to combine these small classifiers in an attempt to boost the accuracy of the composite classifier.

Formal interpretation of the derived training set. Passing all the instances through a set of component classifiers results in a re-representation of the instances. This procedure also partitions the instance space according to the set of classifiers. Formally, the component classifiers induce an equivalence relation on the instance space. Two instances are equivalent modulo a particular set of classifiers if and only if each classifier in the set makes the same prediction when given the two instances as input.

Definition: Let $\mathcal{C}$ be a set of classifiers on a data set T_0. Let $x_i, x_j \in T_0$. Define $x_i \equiv x_j$ iff $C(x_i) = C(x_j) \, \forall C \in \mathcal{C}$.

This relation is an equivalence relation on the training set instances, since it is reflexive, symmetric and transitive. Thus the derived training set is analogous to a *quotient space* $T_0/\equiv$ induced by this relation on the training set T_0. Intuitively, the space has been partitioned according to the perspective afforded the component classifiers. So, informally, the effect of creating the derived training set is to eliminate the details of the instances, and to use only the features supplied by the component classifier predictions. In many algebraic and topological settings the canonical map from an element to the equivalence class to which it belongs is a structure-preserving homomorphism. While the idea has not been formalized for our setting, future work may formalize the intuitive notion that the class structure of a set of instances is preserved by the canonical map from the original instances to the derived representation induced by a set of component classifiers.

Searching the space of nearest neighbor classifiers. The approaches to k-nearest neighbor classifier construction that are presented in this proposal effect a search of a low-dimensional subspace of the space of k-nearest neighbor classifiers. A k-nearest neighbor classifier may be represented as a 4-tuple: $< P, k, d, \theta >$ where

P is a set of prototypes,
k is the number of nearest neighbors to consider,
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.

In this research we vary P, the set of prototypes, to construct classifiers[3]. However, the prototype set is only one component of a k-nearest neighbor classifier. The other "degrees of freedom" could be exploited as well, including varying k and varying the distance metric, d. Also the rule θ that selects the class of an unseen instance from the class of a set of nearest neighbors could be varied. Of course, the default rule is the majority (plurality) rule. Other rules are available, as well. For example, combining nearest neighbors on the basis of a minority rule ("predict a class that receives the fewest, non-zero number of votes from the k nearest neighbors") is also a possibility. Since our objective is to find classifiers whose errors are complementary, a nearest neighbor classifier that applies a minority (or similar) rule may be combined effectively with classifiers that apply the majority rule. The dot product of the error vectors of two classifiers gives one measure of the orthogonality of two classifiers, as defined in Section 5.3.3.

Thus the research we propose can be viewed from several perspectives, in which it extends or provides an analogy to previous research. In the next section we outline several assumptions that we make in order to constrain the research task we face.

3.8 Assumptions

There are several assumptions that we make in this research. In this section we discuss the reasons for making several of them.

No strong domain theory. One of the 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. More sophisticated approaches

[3]Where both k and P should be specified explicitly, the term "$k-P$-nearest neighbor classifier" might usefully refer to a k-nearest neighbor classifier that incorporates P, a set of prototypes.

to these searches can be applied where a model of the domain (and of the available classifiers in that domain) is available, such as in the Integrated Processing and Understanding of Signals (IPUS) architecture of Lesser [Lesser *et al.*, 1994], or in the knowledge-based feature generation of Callan [1993]. On the other hand, our assumption makes the framework we develop applicable to situations where domain knowledge is not available to bias classifier search.

Classifiers from a single model class. Our empirical work will assume that one particular type of classifier is used for the component classifiers whose predictions are being combined. While other research has focused on combining different types of classifiers, the focus in this proposal is on the combination of a collection of homogeneous classifiers, all of which are nearest neighbor classifiers.

We adopt this constraint for a variety of reasons:

1. Nearest neighbor algorithms are prototypical examples of instance-based classifiers, which have been shown to be effective in a variety of domains over a 50-year period. Since there are few examples of stacking component nearest neighbor classifiers in the research literature, studying the stacking of nearest neighbor classifiers will help fill a gap in the evolution of knowledge in this area.

2. Where a set of component classifiers use the same internal concept representation, the potential exists for combining these concept representations, rather than simply combining the class labels output by each classifier. For example, a combining classifier might attempt to base its class prediction on the rankings of the neighbors maintained 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 internal concept representations, at least not nearly as easily. Incidentally, this observation opens up the research question "How much knowledge should a combining classifier have of the operation of the component classifiers?"

3. Nearest neighbor classifiers have been characterized by Breiman as *stable* classifiers, meaning that a small change in the training set only leads to small changes in the hypotheses that are generated by the classifier [Breiman, 1994]. He observes that instability can lead to large generalization errors and large variance, and these characteristics have not been shown to be desirable for component classifiers.

4. It is desirable to keep the space of classifiers to be searched as small as possible, while observing the constraint that the space must contain classifiers that yield high accuracy, once their predictions are combined appropriately. It has not been shown that it is necessary to combine classifiers of different types in order to achieve high accuracy. In fact, it has been observed by Wolpert[4] that Breiman achieved significant improvement even though the classifiers he combined were quite similar to one another [Breiman, 1992]. We attempt to show that while we limit the search space of learning algorithms we can still achieve very good generalization accuracy on a variety of problems.

No artificial prototypes. This research also deliberately avoids the creation of artificial instances because such instances may be unrealistic in the sense that they do not observe implicit constraints that are present in real data. While such artificial data can be useful as prototypes to make class predictions, they may not have the *imprimatur* of real instances in their inherent explanatory power. An analogy can be made to the citation of cases in legal practice: while it may be instructive and probative to cite a hypothetical case, only *bona fide* cases may be cited in a legal brief as an authority for a proposition of law.

Using artificial prototypes would also require solution of the problem of how to create prototypes where some of the features are symbolic or discrete, since the default construction of artificial prototypes takes an average of feature values. For

[4]Personal communication.

example, where a binary feature vector representation is used, the concept of an average instance also probably would be inapplicable.

3.9 Conclusion

This chapter has introduced the problems of classifier composition and the solution paths we propose. We have shown how a composite system that combines a set of small nearest neighbor classifiers may realize our goals of an efficient and accurate classifier. Four general techniques of component classifier construction have been proposed: random sampling, selecting the most accurate of a random sample, error orthogonality and inconsistency reduction. These techniques are discussed in detail in Chapter 5. In the next chapter, Chapter 4, we discuss algorithms for creating a single, independent nearest neighbor classifier as a prelude to combining several of them. In particular we show how search algorithms with a random basis can be used on some data sets to find small prototype sets with good classification accuracy.

Chapter 4

Constructing a Nearest Neighbor Classifier

4.1 Chapter Organization

This chapter is the first of two devoted to specific algorithms to select prototypes and to combine classifiers. Our objective in this chapter is to demonstrate that simple sampling and search methods are adequate to create independent nearest neighbor classifiers with good classification accuracy. This chapter provides evidence that sampling and search can yield component classifiers of sufficiently good accuracy to justify their use in composite classifiers that use voting or other combining algorithms that depend directly on the accuracy of the components. However, classification accuracy is only one criterion we shall apply in the construction of composite classifiers generally, and low-accuracy classifiers may nevertheless serve as useful components.

4.2 Introduction

The classical nearest neighbor algorithm has a probably deserved reputation for being computationally expensive. Run-time costs for the basic algorithm are high: in order to determine the class of a new instance, its similarity to every instance in memory is assessed. Retaining all instances in primary memory also entails storage costs. Our goal in this chapter is to demonstrate how two algorithms that rely on random sampling and local search can reduce the cost of nearest neighbor classification at run-time by reducing the number of prototypes retained. While a classical nearest neighbor algorithm treats all instances as prototypes, we show that nearest neighbor classification accuracy on out-of-sample data from four standard data sets may be maintained or even increased by selecting only a small handful of instances as prototypes. In order to reduce costs in space and time further, we also show how local search can be applied to limit the features used in the nearest neighbor computation.

Many previous approaches to selecting prototypes are instance-filtering techniques, where each member of the data set is examined in turn and some screen is used to sift the elements that should be retained in the emerging concept description. Since our goals are to decrease as far as possible the number of prototypes used and to select prototypes that together classify well, our approach starts from an *a priori* specification of prototype set size and tries to construct an accurate prototype set of that cardinality. This approach has the obvious advantage of forcing the examination of very small sets of prototypes. It has the potential disadvantage of requiring a large amount of search to locate them. In addition, clamping the number of prototypes limits the difficulty of the problem we address. While we do not pursue the question of how many prototypes to include, we do appeal to a clustering index in Section 4.7 that may be applied to this important problem.

Two algorithms are applied to select prototypes and features used in a nearest neighbor algorithm: (1) a Monte Carlo technique, which chooses the most accurate of a sample of random prototype sets; and (2) a random mutation hill climbing algorithm (*e.g.,* [Mitchell and Holland, 1993]), which searches for sets of prototypes with demonstrably good classification power. In previous work we used a genetic algorithm to find prototypes [Skalak, 1993], which applied greater computational resources to prototype selection, with comparable results. This chapter follows the theme of using adaptive search techniques to find sets of prototypes, but uses somewhat less computationally intensive algorithms to locate them.

4.3 The Nearest Neighbor Algorithm Applied

To determine the classification accuracy of a set of prototypes, a 1-nearest neighbor classification algorithm is used [Duda and Hart, 1973]. The similarity function used in this nearest neighbor computation is straightforward and relies on equally-weighted features.

Pre-processing. Data are pre-processed (1) to scale values and (2) to instantiate any missing feature values[1].

1. Scaling. All feature values are linearly scaled from 0 to 100. Extreme numeric feature values are squashed by giving a scaled value of 0 (100) to any raw value that is less (greater) than three standard deviations from the mean of that feature computed across all 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 previously incorporated a similar data pre-processing method [Cognitive Systems, Inc., 1992].

2. Missing Values. In the pre-processing step, missing feature values are (naively) instantiated with the median value of that feature across all instances.

Similarity Computation. To compute the similarity distance between two scaled instances, we use the Manhattan ("city block" or l_1) distance metric. The prototype with the smallest such similarity distance to a test instance is its 1-nearest neighbor. 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 1-nearest neighbor taken from the prototype set.

Our algorithm also incorporates, for unordered, symbolic values, the Stanfill and Waltz *value difference metric* [Stanfill and Waltz, 1986]. The value difference metric uses the relative frequency of values across classes to determine the distance between each pair of symbolic values that a feature can assume. Two symbolic values are close if for each class they appear in instances in that class with the approximately the same relative frequency. A slight variation on the metric was subsequently used by Cost and Salzberg [Cost and Salzberg, 1993].

[1] Of the four data sets we use in this chapter, only the Cleveland Heart Disease has any missing values; six feature values are absent.

We introduce a Monte Carlo method (MC1) and two applications of random mutation hill climbing algorithms (RMHC-P and RMHC-PF1). Finally, we offer evidence that the degree of clustering of the data set is a factor in determining how well our simple sampling algorithm will work.

4.4 The Algorithms

4.4.1 Baseline Storage Requirements and Classification Accuracy

Four databases from the UCI machine learning repository [Murphy and Aha, 1994] were used: Iris, Cleveland Heart Disease (binary classes), Breast Cancer Ljubljana, and Soybean (small database). All but the Soybean database were chosen in part because results using various algorithms were compiled from the research literature by Holte [1993], providing convenient touchstones for comparison. As Holte records, these databases vary along such features as size, the accuracy of a classification rule that selects the most frequent class, whether missing values are extant, whether continuous features are present, the distribution of the number of distinct values of discrete attributes, and the total number of attribute values. In Section 6.3 we propose selecting a number of datasets that represent a variety of tasks in the dissertation research to be completed.

As a basis for comparison for the results we will present in this chapter, we computed the baseline classification accuracy of the nearest neighbor algorithm, using all the training cases as prototypes and all the features, and five-fold cross validation. (Folds of equal size were used in the five-fold cross validation, necessitating that a residue of fewer than five instances might be left out of every fold.) The average accuracy over the five folds is given in Table 4.1. For general reference, since C4.5 [Quinlan, 1993] is a benchmark learning algorithm, we also include in Table 4.1 classification accuracies on the four data sets from our own five-fold cross validation runs using pruned trees generated by C4.5 with its default option settings.

Table 4.1. Storage requirements (with number of instances in each data set) and classification accuracy computed using five-fold cross validation with the 1-nearest neighbor algorithm used in this chapter and pruned trees generated by C4.5. The symbol "*" denotes statistical significance at the 0.05 confidence level. A two-sample t-test for statistical significance assuming equal population variances was used.

Data Set	**Storage**	**1-NN**	**C4.5**
Iris	100% (150)	93.3%	93.3%
Cleveland	100% (303)	74.3%	71.6%
Breast Cancer	100% (286)	65.6%	72.4%*
Soybean	100% (47)	100.0%	95.6%

In general, direct comparison with published results may be improvident in that different validation techniques, similarity metrics, and values of k of the k-nearest neighbor algorithms, and a different cross-validation partition almost surely will have been used. In this research, we did not try to optimize k or experiment with different similarity metrics.

4.4.2 Monte Carlo (MC1)

As a general matter, Monte Carlo methods provide approximate solutions to mathematical and scientific problems through repeated stochastic trials. The results of independent trials are combined in some fashion, usually averaged [Sobol', 1974]. The method has been applied to many problems in such domains as numerical integration, statistical physics, quality control and particle physics.

The algorithm described in this section, called MC1, is a simple application of repeated sampling of the data set, where sampling is done with replacement. The algorithm is simple. It takes three input parameters: k (k-nearest neighbors), m (the number of prototypes, which is the sample size), and n (the number of samples taken). These parameters are fixed in advance. For all the experiments presented in this chapter, $k = 1$ and $n = 100$. We choose $m = 3$ for all but the Soybean data set, where $m = 4$, since there are four classes.

The Monte Carlo MC1 classification procedure can be summarized as follows. We assume that the data set has been divided for experimental purposes into a training set and test set.

1. Select n random samples, each sample with replacement, of m instances from the training set.

2. For each sample, compute its classification accuracy on the training set using a 1-nearest neighbor algorithm.

3. Select the sample(s) with the highest classification accuracy on the training set.

4. Classify the test set using as prototypes the samples with the highest classification accuracy on the training set. If more than one sample has the same highest classification accuracy on the training set, select a classification randomly from those given by these best-performing samples[2].

In a real application, all previously seen instances would be used as training instances. We report in Table 4.2 the storage requirements (the percentage of members of the data set that were retained) and the classification accuracy of the MC1 algorithm, applying five-fold cross validation.

One of the many questions that surfaces in the wake of these results is the effect of the number of samples of prototype sets on test set accuracy for this algorithm. While more experimentation would be required to give a full answer to this question, to arrive at a preliminary indication we performed a 10-fold cross validation experiment in which we determined the accuracy on the test set of the prototype set having the highest accuracy on the training set after 10, 100 and 1000 samples of prototype sets. We performed this experiment for three data sets, omitting soybean because of the high accuracies achieved on that data. The number of prototypes in each prototype

[2]We report average accuracy on the test set of the prototype sets that display the highest predictive accuracy on the training set.

Table 4.2. Storage requirements for the 1-nearest neighbor and MC1 algorithms, average MC1 classification accuracy and average baseline 1-nearest neighbor classification accuracy using five-fold cross validation.

Data Set	**1-NN Storage**	**MC1 Storage**	**1-NN**	**MC1**
Iris	100%	2.0%	93.3%	93.5%
Cleveland	100%	1.0%	74.3%	80.7%
Breast Cancer	100%	1.0%	65.6%	72.6%†
Soybean	100%	8.5%	100.0%	99.1%

set (3) and other initial parameters are the same for this algorithm as in the previous experiment. The average accuracy on the test set is given in Table 4.3.

Table 4.3. Effect on test set accuracy (average percent correct) of number of prototype sets sampled.

Data Set	**10**	**100**	**1000**
Iris	91.3%	95.3%	95.3%
Cleveland	80.0%	79.7%	82.0%
Breast Cancer	71.4%	71.4%	71.4%

These experiments show that MC1, a very simple approach based on random sampling, does quite well on these four data sets, with a large reduction in storage. In each of the classification accuracy tables we present, the symbol "†" denotes statistically significant improvement over the baseline 1-nearest neighbor algorithm (1-NN) at the 0.1 confidence level; "*", significance at the 0.05 confidence level. A two-sample t-test for statistical significance assuming equal population variances is used. In Section 4.7 we try to characterize one of the factors that will determine when random sampling will provide good accuracy. An approach to prototype selection based on random mutation search is described next.

4.4.3 Random Mutation Hill Climbing

4.4.3.1 The Algorithm (RMHC)

Random mutation hill climbing is a local search method that has a stochastic component [Papadimitriou and Steiglitz, 1982]. The basic random mutation hill climbing algorithm (RMHC) is as described by Mitchell and Holland [Mitchell and Holland, 1993]:

1. Choose a binary string at random. Call this string *best-evaluated*.

2. Mutate a bit chosen at random in *best-evaluated*.

3. Compute the fitness of the mutated string. If the fitness is strictly greater than the fitness of *best-evaluated*, then set *best-evaluated* to the mutated string.

4. If the maximum number of iterations have been performed return *best-evaluated*; otherwise, go to Step 2.

The general approach is to use a bit string to represent a set of prototypes, and in some experiments, a collection of features. The intuitive search mechanism is that the mutation of the bit vector changes the selection of instances in the prototype set or toggles the inclusion or exclusion of a feature from the nearest neighbor computation. The fitness function used for all the RMHC experiments is the predictive accuracy on the training data of a set of prototypes (and features) using the 1-nearest neighbor classification algorithm described in Section 1.

4.4.3.2 Search for Prototype Sets (RMHC-P)

For this experiment, the bit vector encodes a set of m prototypes in a straightforward way. Each prototype set is encoded as a binary string, which is conceptually divided into m substrings, one for each of the m prototypes, where each substring encodes an index into the cases stored as an array. The length of the binary string encoding a prototype set is the number of bits required to represent the largest index of a case in the data set, multiplied by the number of prototypes, $\lceil log_2 r \rceil \cdot m$ where r is the number of cases in the data set. So, for example, the number of bits used to encode a set of three Iris prototypes was $\lceil log_2 150 \rceil \cdot 3 = 24$.

The RMHC algorithm was applied to this representation, searching for a set of three (four for the Soybean data set) prototypes with superior predictive power. The fitness function was the classification accuracy on the training set of a 1-nearest neighbor classifier that used each set of prototypes as reference instances. In the experiments reported here the algorithm was run for 100 mutations[3]. The results after only 100 mutations — a very small number for this type of approach — are presented for two reasons. Informally, we did not observe an increase in classification accuracy in many experiments by running the algorithm longer, although more experimentation is required to establish this result. Second, such a small amount of search supports a *sub rosa* hypothesis of this chapter — that small sets of prototypes with good classification accuracy are denser in the space of sets of prototypes than might be expected for these data sets, since little search is required to locate them.

Further, the accuracy of the final prototype set returned by the algorithm does appear to be better than the random prototype set used as the starting point. A random starting prototype set yielded an average accuracy (across the five partitions) on the test set of 68.0% for the Iris data set, 54.7% for the Cleveland data, 61.4% for the Breast Cancer data, and 66.7% for the Soybean data. Average classification accuracy for the final prototype set and storage requirements for five-fold cross validation are given in Table 4.4. The storage requirements for the baseline nearest neighbor algorithm are 100% of the instances for all the experiments reported in this chapter.

[3] To accelerate the algorithm, the calculated fitness of each prototype set was cached by memoizing [Abelson *et al.*, 1985] the evaluation function so that the predictive accuracy of a set of prototypes (and features) need only be computed once for each fold of the cross validation. Nonetheless, retrievals of previously cached evaluations are counted in the number evaluations reported.

Table 4.4. Storage requirements for the prototypes found by RMHC-P, average classification accuracy for the prototypes selected by RMHC-P, and average baseline 1-nearest neighbor classification accuracy.

Database	**Storage**	**RMHC-P**	**1-NN**
Iris	2.0%	93.3%	93.3%
Cleveland	1.0%	82.3%*	74.3%
Breast Cancer	1.0%	70.9%	65.6%
Soybean	8.5%	97.8%	100.0%

The results show a comparable or improved classification accuracy of the RMHC-P algorithm over a nearest neighbor algorithm using all the training instances as prototypes. In particular, the results on the Cleveland database appear to be better than previously published results with only a very small percentage of stored instances used.

4.5 Search for Prototype and Feature Sets (RMHC-PF1)

Finally, experiments were performed to determine if the RMHC algorithm could select features as well as prototypes. Further reduction in computational costs would result from a reduction in the number of features that have to be considered in the nearest neighbor similarity computation. We used RMHC-PF1 to select prototypes and features simultaneously, using a representation that records both the prototypes and features in a single vector.

The original numbers of features in the datasets we consider are given in Table 4.7, and these numbers are quite small, varying from 4 to 36. We do not know how well this algorithm would perform for instances with large numbers of features. For example, in an information retrieval system applying a vector space model of retrieval, a document is usually represented as a sparse binary vector, with 1's representing the appearance of a word (from some fixed dictionary of thousands of terms) in that document, and 0's representing the absence of a word. This algorithm has not been tested on such datasets. Nevertheless, reducing the features stored for each instance is a laudable goal, since it reduces costs in both time and space, reducing on-line storage and the computation that must be done to compute the distance between any two instances. Finally, in view of the fact that some UCI datasets sometimes have been criticized as including only engineered features that are relevant (to the extent that such engineering can be done reliably, however, it is a good thing), it is somewhat surprising that feature reductions can be achieved at all from such a small base of original features.

To use RMHC to select features, we used a simple characteristic function bit vector representation, one bit for each feature used in the instance representation. The ith bit records whether to use the corresponding ith feature from some fixed presentation of the features: 1 to include the feature in the similarity distance computation, 0 to exclude it. Thus, for the Cleveland database, there are 13 features, and a 13-bit vector represents the features.

The bit vectors used in the previous experiments for prototypes and the features bit vector were concatenated in this representation. At each iteration only one bit was mutated, and it was left to the random bit mutation procedure, which with uniform probability randomly selects a bit to toggle, whether that bit fell within the "prototypes sub-vector" or the "features sub-vector." We did not attempt to alter any bias stemming from the different relative lengths of the prototype set vectors and feature vectors. RMHC-PF1 used a fitness function that classified the training set using the encoded prototypes and only taking into account the features that are specified by the bit vector.

For consistency of experimental presentation, the algorithm was again run for 100 evaluations only, notwithstanding the increased size of the search space, starting with a random set of features and prototypes. The random starting prototype and features set yielded an average accuracy (across the five partitions) on the test set of 51.3% for the Iris data set, 61.6% for the Cleveland data, 55.4% for the Breast Cancer data, and 51.1% for the Soybean data. Classification accuracy and storage requirements for the five-fold cross validation are given in Table 4.6.

If a data set has P instances, each containing F features, and the RMHC-PF1 algorithm yields p prototypes and f features, the storage requirements reported are pf/PF. For example, the RMHCP-PF1 algorithm retrieves $p = 3$ prototypes and an average, across the five folds, of $f = 2.4$ features; there are $P = 150$ original instances and $F = 4$ original features. The total storage costs are therefore given as $pf/PF = (3 \cdot 2.4)/(150 \cdot 4) = 0.012$ See Table 4.5 for the other computations. As always, the 1-NN algorithm requires 100% storage.

Table 4.5. Computation of average storage requirements for RMHC-PF1

Database	**Selected Prototypes**	**Selected Features**	**Total Instances**	**Total Features**	**RMHC-PF1 Storage**
Iris	3	2.4	150	4	1.2%
Cleveland	3	7.6	303	13	0.6%
Breast Cancer	3	4.8	286	9	0.6%
Soybean	4	16.4	47	36	3.9%

Table 4.6. Storage requirements and average classification accuracy for the selection of prototypes and features by RMHC-PF1, with average 1-nearest neighbor baseline classification accuracy.

Database	**Storage**	**RMHC-PF1**	**1-NN**
Iris	1.2%	94.7%	93.3%
Cleveland	0.6%	80.7%†	74.3%
Breast Cancer	0.6%	72.3%*	65.6%
Soybean	3.9%	97.8%	100.0%

As shown in Table 4.7, approximately half the total number of features were used in general and so the reduction in features alone cuts run-time storage costs in half. For example, an average, across the five folds, of 2.4 features out of 4 were used for Iris classification. *Petal-width* and *petal-length* appear to be useful predictive features, since they were selected as features in five and four partitions, respectively. *Sepal-length* is apparently not useful, since it was not selected in any partition. An average of 7.6 features of 13 were used for classification in the Cleveland data set. The *ca* feature was used in all five partitions; *cp*, *exang*, and *thal* were applied in four; *restecg* in none. The features for the Cleveland data set are described in Appendix C. Descriptions of these features may be found in the UCI repository [Murphy and Aha, 1994]

Table 4.7. Number of features in the original instance representation and average number features selected by RMHC-PF1.

Database	**Total Features**	**RMHC-PF1 Features**	**Percent Features Retained**
Iris	4	2.4	60%
Cleveland	13	7.6	61%
Breast Cancer	9	4.8	53%
Soybean	36	16.4	46%

4.6 Discussion

Table 4.8 summarizes the mean predictive accuracy of the preceding experiments.

Except for the small Soybean data set, the storage requirements for all of the algorithms were about 1%-2% of the training instances, except for the baseline nearest neighbor algorithm, which used 100% of the training examples. The general lesson is that a reduction in storage costs of one or two orders of magnitude from a standard nearest neighbor algorithm that uses all instances has been achieved on some of these data sets together with a statistically significant increase in computational accuracy.

Table 4.8. Summary of average classification accuracy (% correct) from five-fold cross validation for the experiments presented in this chapter to select prototypes and features. Storage requirements, in percentage of the data set are given in parentheses. The symbol "†" denotes statistically significant improvement over the baseline 1-nearest neighbor algorithm (1-NN) at the 0.1 confidence level; "*", significance at the 0.05 confidence level. A two-sample t-test for statistical significance assuming equal population variances was used.

Database	**1-NN**	**MC1**	**RMHC-P**	**RMHC-PF1**
Iris	93.3	93.5 (2.0)	93.3 (2.0)	94.7 (1.2)
Cleveland	74.3	80.7 (1.0)	82.3* (1.0)	80.7†(0.6)
Breast Cancer	65.6	72.6†(1.0)	70.9 (1.0)	72.3* (0.6)
Soybean	100.0	99.1 (8.5)	97.8 (8.5)	97.8 (3.9)

While some of the nominally better results may not be statistically significant[4], these experiments at least show that using three or four prototypes and possibly a proper subset of the features performs statistically as well as using the entire training set and all the features. Thus the accuracies of the algorithms presented in this chapter are statistically comparable to a standard nearest neighbor approach with much smaller computational expense.

4.7 A Measure of Clustering

It is clear that such naive sampling techniques will not always work, although their limits need to be determined experimentally and theoretically. (One avenue of research is to determine whether random sampling of sets of prototypes can itself provide a measure of the predictive complexity of a database.) The success of a sampling algorithm appears to depend partly on the distribution of instances in the data set and the geometric structure of the concepts to be learned.

[4]In interpreting the significance results in the table, note that the t-test considers the mean and variance of the underlying cross validation data, but that only the mean percentage accuracy is reported here. In general, MC1 displayed higher variance than the other algorithms. A test for the analysis of variance (single factor) for each data set also reveals that we cannot reject at the 0.05 confidence level the null hypothesis that all of the predictive accuracy means are equal.

In particular, one possible explanation for the successful results from sampling presented in this chapter is that the data sets used exhibit well-defined, widely spaced classes[5] in feature space, classes that exhibit a high degree of "internal coherence" and "external isolation." The intuition is that such an ideal separation of classes moots the selection of a prototype, since any instance in an isolated class may give perfect accuracy via a nearest neighbor algorithm.

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

$$[trace\, B/(m-1)]/[trace\, W/(n-m)]$$

where n is the total number of data instances and m is the number of clusters in a possible clustering[6]. 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]. See Appendix A for a description of these matrices.

We performed a set of experiments to determine the effect of class isolation and cohesion, measured by the Calinski-Harabasz Index, on the performance of the Monte Carlo (MC1) sampling algorithm. We regard the instances of each class as a cluster and have applied this index to determine how well the classes are separated within each data set. Our hypothesis was that as the Calinski-Harabasz Index increased,

[5] We thank Paul Utgoff for this suggestion.

[6] The *trace* of a square matrix is the sum of its diagonal elements.

entailing greater class cohesion and external isolation, the performance of MC1 would also increase.

For each of the four data sets, we performed a 10-fold cross validation. In each ply, the following procedure was used, where 90% of the data set was used for training and 10% used for testing.

1. Run MC1 algorithm on the training set and determine the generalization accuracy of the classifier formed with the prototypes computed by MC1 on the corresponding test set.

2. For each pair of classes, compute the Calinski-Harabasz Index using the instances in the test set and treating the instances in each class as a cluster.

3. Take the minimum of the Index values over all pairs of classes.

4. Create a data point, an ordered pair whose abscissa is the minimum Calinski-Harabasz index computed in Step 3, and whose ordinate is the generalization accuracy on the test set.

This procedure allows us to determine the relationship between the minimum Calinski-Harabasz Index (taken over all the pairs of classes in a test set) and the classification accuracy on out-of-sample test sets[7]. The minimum Index was used as the independent variable under the hypothesis that the worst separation between a pair of classes would dominate the classification accuracy. Following this procedure in each of the 10 plies gives 10 data points. The results are graphed in Figures 4.1, 4.2 and 4.3.

From the standpoint of using the Index to *predict* the suitability of a sampling technique for a given data set, the Index value on the training set might better reflect the "true" clustering displayed by the entire data set. Experimental results that apply the Index to the training set have not been conclusive, however.

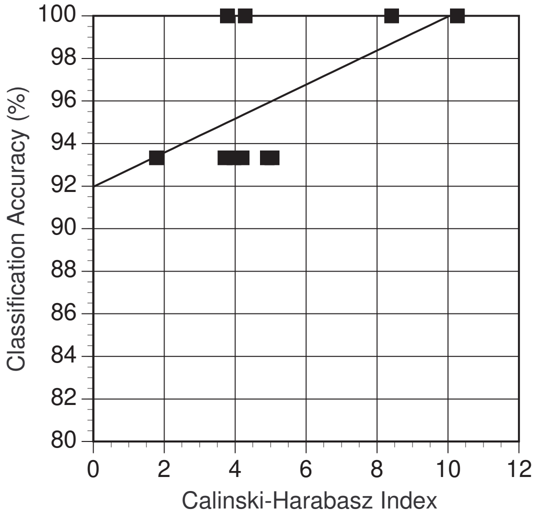

Figure 4.1. Classification accuracy vs. Calinski-Harabasz Index on Iris data

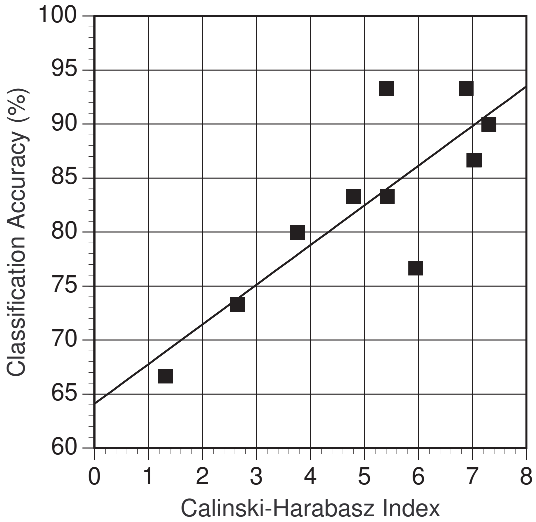

Figure 4.2. Classification accuracy vs. Calinski-Harabasz Index on Cleveland Heart Disease data

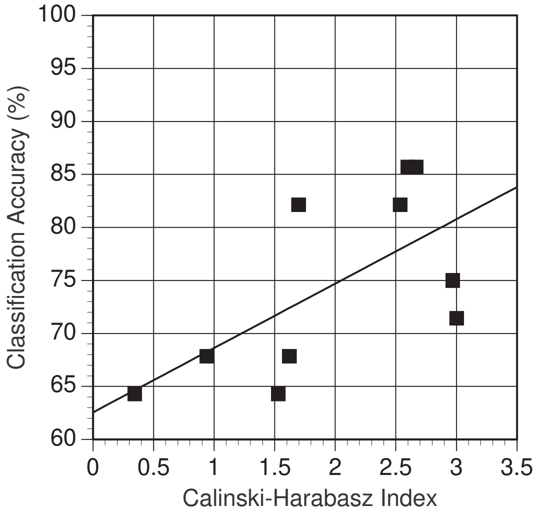

Figure 4.3. Classification accuracy vs. Calinski-Harabasz Index on Breast Cancer data

The results show a clear tendency for MC1 to perform better when the cluster index is higher (good class separation) and worse when the cluster index is lower (poor separation). Tests for significance of the regression trendlines were confirmed by an analysis of variance F-test, where all were found significant at the 0.05 confidence level (Iris: $F = 6.2$; Cleveland: $F = 17.6$; Breast Cancer: $F = 5.4$). While the results are not conclusive, they do present a basis for additional experiments to determine the range of applicability of Monte Carlo approaches.

4.8 Conclusion

We have found these results surprising. Using very simple stochastic algorithms these experiments show that significant reductions in storage of cases and features can be achieved on the data sets examined without decreasing nearest neighbor classification accuracy, and in some instances actually improving it. This chapter also has provided evidence for the hypothesis that the average accuracy of a simple method of finding prototypes by sampling increases with the internal coherence and

[7] Due to the uniformly high performance on the Soybean data, we omit the analysis of that data set.

external isolation of the classified data, as measured by a classical clustering index developed by Calinski and Harabasz.

Chapter 5

Combining Nearest Neighbor Classifiers

5.1 Chapter Organization

In the previous chapter, we discussed the application of sampling and search methods to construct a single nearest neighbor classifier. In this chapter we begin to investigate how to select classifiers whose predictions are to be combined and how to combine nearest neighbor classifiers to get a more accurate composite classifier. Experimental evidence is presented to support the following contention: composite nearest neighbor classifiers composed of a small set of nearest neighbor classifiers each of which incorporates only a small number of prototypes demonstrate better generalization accuracy on several standard data sets than a nearest neighbor classifier that uses all instances as prototypes.

In the specific instantiations of this architecture we will investigate here, the combining classifier applies one of three algorithms:

- plurality vote
- k-nearest neighbor
- ID3.

In the following section we discuss the reasons for selecting these particular combining algorithms. The component classifiers are k-nearest neighbor classifiers in this instantiation.

We now turn to a consideration of the three subproblems involved in the construction of a composite classifier discussed in Chapter 1: selection of a combining classifier (Problem 1), selection of component classifiers (Problem 2), and the integrated selection of combining and component classifiers (Problem 3).

5.2 Selection of a Combining Classifier

In this first problem, we assume that the component classifiers have been fixed in advance and that the goal is to find a superior way to combine them. In this section we give the details of how to train and apply voting, nearest neighbor and ID3 algorithms in a combining classifier.

The hypothesis that we want to test experimentally in this section is:

Hypothesis. On the tested data sets, either of two classical learning algorithms used as a combining classifier yields better classification accuracy than voting used as a combining algorithm.

The selection of a combining classifier might be made by analogy to the ways that a committee of (human) individuals can make a group decision. A panel of appellate judges is one example of a committee whose decisions have to be aggregated in order to render a judgment. In addition to the usual approach of using persuasion and arguing the merits of each potential decision, there are at least three ways for a committee to render a decision given the opinions of the individuals on the committee, at least that might satisfy some elementary sense of fairness.

1. Vote and take the decision with the most votes.

2. Appeal to precedent and make the decision that was made the last time the committee members held their present opinions as to which decision is best.

3. Determine which subset of members are best at making the decision and let them decide.

These three approaches serve as analogies for our choices of combining classifiers: voting, nearest neighbor and ID3. These off-the-shelf algorithms have been selected deliberately to show that once an appropriate derived training set has been created, a custom algorithm is not needed to generalize from that new training set. As we discussed in Chapter 2, voting is a default method, but will be unable to generalize situations where a majority of the classifiers give the wrong answer. Both ID3 and a k-nearest neighbor algorithm should be able to learn from situations where a majority of the component algorithms are wrong. ID3 and k-nearest neighbor algorithms demonstrate different strengths in our context. ID3 should be able to recognize when to output the prediction of a particular component classifier, or a prediction based on a small subset of them. Using a nearest neighbor combining classifier gives a uniform framework for the composite classifier by incorporating nearest neighbor classifiers at both levels, but the benefits of this uniformity remain to be demonstrated. Nearest neighbor algorithms may give strong performance where the component classifiers are equally accurate, that is, where all the derived features are equally relevant. ID3 may prevail where the classifiers vary in accuracy, since it need not take all the classifier predictions into consideration.

Before we describe how to train and apply these classifiers, we first mention a characteristic of a derived training set that is not typically encountered in raw training sets (at least not in the UCI repository): inconsistent instances, instances whose feature values are the same but whose class labels are different. The component classifiers compress the data and if the compression is too severe, the presence of inconsistent derived instances is a potential hazard. (As we shall see in Section 5.3.2, attempting to limit excessive compression by the component classifiers at the derived representation level leads to a heuristic method for selecting component classifiers.)

For each of the three combining algorithms we apply, we describe how they have been adapted to deal with inconsistent instances, and how they are trained and applied in a composite classifier. The initial setup of the derived feature set that forms the training set for the combining classifier is the same for the k-nearest neighbor and ID3 algorithms and was described Section 3.3, where pseudocode for the training algorithm was provided. No training is required for the voting combining algorithm.

Voting. In the application phase, a test instance is input to each of the individual classifiers, and each classifier outputs a class prediction. The class predicted by the greatest number of component classifiers wins and constitutes the final prediction of the composite classifier. Ties among component classifier predictions are broken randomly.

k-Nearest Neighbor. In the application phase, a test instance is input to each classifier and a vector of class predictions is formed, as in the training phase. Nearest neighbors in the derived case base to the test instance are computed using a distance function that counts the number of class predictions that are different in the two vectors. Essentially, this is a Hamming distance metric applied to symbolic (rather than binary) vectors.

The implementation of any k-nearest neighbor algorithm must incorporate a policy for determining the instances in a neighborhood where more than k instances can legitimately be included in the neighborhood. This policy is invoked when there are many instances on the boundary of a neighborhood. These boundary instances are those that are as close as the farthest of any set of k neighbors. The policy of the k-nearest neighbor combining algorithm we apply is to include all these boundary instances, that is, all the instances whose distance to a test case is the same as that of the k-th nearest neighbor.

In general, the predicted class is given by plurality vote of the classes of instances in a neighborhood. Ties among these instance votes are broken randomly. The default value of k in these experiments is $k = 1$.

ID3. The set of derived vectors is then input to the ID3 algorithm to train a decision tree. Internal nodes in the tree branch on the prediction of a particular component classifier, and therefore branches correspond to the predictions of a particular classifier. The gain ratio feature selection criterion is used. In the application phase, a test instance is classified by each of the component classifiers. The derived instance is then input to the decision tree combining algorithm, which outputs a prediction.

In the implementation of ID3 used here[1], a plurality vote of instances at each leaf containing inconsistent instances is used to make class predictions for test instances that fall to each such leaf.

In the next section we describe an experiment that shows for several datasets that composite algorithms can outperform the baseline nearest neighbor algorithm and compares the combining classifiers we consider.

5.2.1 Experiment

This experiment compares the classification accuracies of the three combining algorithms on several data sets. We use a very simple approach to selecting component classifiers, random selection. In a ten-fold cross validation experiment, in each fold we randomly selected three sets of three prototypes from the training set. Three nearest neighbor classifiers were created using the three sets of prototypes. A composite classifier was built from these three classifiers, using each of the three combining functions. Table 5.1 gives the average accuracy of the composite classifiers across the ten folds.

We compare the accuracy of the three combining algorithms to an "unstacked" baseline classifier, a 1-nearest neighbor algorithm using all the instances as prototypes. The accuracy of the baseline classifier reported in Table 5.1 is different from the 1-nearest neighbor baseline accuracy reported in the experiments in Chapter 4 because five-fold cross validation was used in Chapter 4 and ten-fold cross validation was used in the experiments in this chapter.

Since ID3 and k-nearest neighbor perform comparably, these experiments do not argue in favor of the universal application of any one of the combining algorithms for

[1]Thanks go to Neil Berkman for writing and providing Lisp source code for ID3.

Table 5.1. Comparison of baseline 1-nearest neighbor with the ID3 algorithm, $k-$nearest neighbor and voting, for combining randomly selected component classifiers. Average percent classification accuracy on test data, using ten-fold cross validation. The symbol "†" denotes statistically significant improvement over the baseline 1-nearest neighbor algorithm (NN) at the 0.1 confidence level; "*", significance at the 0.05 confidence level.

Data Set	**Baseline**	**Combining Algorithm**		
	1-NN	**ID3**	***k*-Nearest Neighbor**	**Vote**
Iris	91.3	86.8*	86.2*	82.4*
Breast Cancer	68.9	70.7	70.7	56.8*
Cleveland Heart Disease	76.0	75.5	75.3	66.3*

amalgamating the predictions of small, randomly selected nearest neighbor classifiers. However, they do provide some support for our experimental hypothesis, that voting is not the universally best combining algorithm, although it is the combining method often relied upon as a default [Battiti and Colla, 1994; Littlestone and Warmuth, 1989]. With these preliminary results[2], we turn to our main focus, the search for complementary component classifiers.

5.3 Search for Component Classifiers

In this section, we consider how to select or construct the component classifiers whose decisions are being combined. We rely on three general approaches, each of which is embodied in one or more algorithms. The general hypothesis we wish to demonstrate is:

Hypothesis: A composite classifier can yield greater generalization accuracy than a nearest neighbor algorithm that uses all training instances as prototypes.

We discussed four approaches to component classifier selection in Section 3.4. The algorithms are described in greater detail below. The output of each algorithm is a set of nearest neighbor classifiers to be used as component classifiers. After each of the algorithms has been described, we present an experiment with a factorial design. For three datasets, we show the average generalization accuracy of a composite classifier

[2] Additional datasets will be used in the experiments in the proposed dissertation.

consisting of component classifiers selected by each of the four algorithms we present, and combined using each of the three combining algorithms.

5.3.1 Classifier Selection through Sampling

There are two algorithms that select classifiers through sampling: *Random* and *Best of Random*. The Random selection algorithm takes three inputs: the number of classifiers to be created (n), the number of prototypes to be used in each classifier (p), and a training set (T). In this proposal we assume that each classifier uses the same number of prototypes. This algorithm randomly selects n subsets of instances, with replacement, of cardinality p from T. The sampling is constrained to select at least one prototype from each class exposed in the training set. For each of the n subsets, a nearest neighbor classifier is constructed with that set of instances used as the prototype set. The Manhattan metric is used as a similarity metric.

The Best of Random classifier selection algorithm takes the same set of arguments as the Random selection algorithm, plus one additional input, the number of samples to take (s). Then s samples of p prototypes are sampled with replacement from T. Next, s nearest neighbor classifiers are constructed from these prototype sets. The classification accuracy on the training set T is determined for each of the s classifiers. The n classifiers with the highest training set classification accuracy are output. Ties are broken randomly.

5.3.2 Classifier Selection through Inconsistency Reduction

In this section we describe the Inconsistency Reduction algorithm that adds component classifiers dynamically to reduce the inconsistency of the derived training set. A *set* of instances is inconsistent if there exist two or more instances in the set that have (all) the same feature values but different class labels. Recall that a derived training set is created by inputing each instance in the original training set to all of the component classifiers and re-representing the instance as a vector of class predictions and its known class label. Each component classifier provides a feature for the derived training set.

We use a measure of the inconsistency of a representation to drive a greedy search for a level of representation that will reduce the inconsistency in the derived training set. Beginning with the feature given by an initial component classifier, the representation is expanded to include the features given by another component classifier when the representation exhibits too great a degree of inconsistency (with "too great" currently specified by the user). The expansion of the representation stops when the training set inconsistency falls below the user-specified threshold. If inconsistency in the derived training set remains when the algorithm halts, it will reduce the classification accuracy on the training set, and, possibly, on out-of-sample instances. Thus the algorithm effects a greedy search in a space of representations for a data set. This space of representations is quite limited, however, since the features of each instance representation are merely class labels. Nonetheless, it is a question for future research whether a representation that is appropriate for the learning algorithms we apply can be found within this somewhat impoverished set of representations.

We describe the algorithm in more detail below, but at each stage the inconsistency of the derived training set is reduced by choosing two instances from the original training set whose representations in the derived training set are inconsistent. These inconsistent instances are used as prototypes for a new classifier that gives rise to another feature in the derived representation. The algorithm attempts to choose the instances in a way that will help to decrease the inconsistency.

For example, suppose that x_1 and x_2 are two original training instances with class labels 0 and 1, respectively, and that the current set of component classifiers is $\{C_1, C_2\}$. Suppose the derived representation of x_1 is $< 0 \quad 1 \quad 0 >$: that is, component classifier C_1 outputs 0 on input x_1, component classifier C_2 outputs 1 on input x_1, and the class of x_1 is 0, as we assumed. Suppose similarly that the derived representation of x_2 is $< 0 \quad 1 \quad 1 >$. Now the derived representations of x_1 and x_2, $< 0 \quad 1 \quad 0 >$ and $< 0 \quad 1 \quad 1 >$, are inconsistent. These instances have the same feature values ($< 0 \quad 1 >$), given by the predictions of C_1 and C_2, but different class labels (0 and 1). The Inconsistency Reduction algorithm attempts to create a third classifier, C_3, whose addition to the set of component classifiers will reduce the inconsistency in these two instances. For example, if C_3 output 0 on input x_1, but C_3 output 1 on input x_2, then the inconsistency in these two instances would be removed. The representation of x_1 induced by the three component classifiers would be $< 0 \quad 1 \quad 0 \quad 0 >$, and the derived representation of x_2 would be $< 0 \quad 1 \quad 1 \quad 1 >$. The problem is how to construct component classifier C_3 by choosing suitable prototypes for it. For example, the inconsistent instances themselves (x_1 and x_2) may be used as two of the prototypes in C_3, which is the current approach of the Inconsistency Reduction algorithm.

While low inconsistency in the derived training set appears to be a prerequisite for good classification accuracy in a composite classifier, we are experimenting with various implementations of the Inconsistency Reduction algorithm. We therefore describe the algorithm in two stages, first in broad outline to highlight the design issues and then in more, albeit provisional, detail. We follow the same two-step description procedure for the Error Orthogonality algorithm to select component classifiers, described in Section 5.3.3.

In broad outline, the steps of the Inconsistency Reduction algorithm are:

1. Create an initial component classifier.

2. Compute the inconsistency of the derived training set.

3. If exit conditions are satisfied, then return the current set of component classifiers.

4. Otherwise, construct a classifier and add it to the component classifier set.

5. Go to step 2.

As this outline shows, there are several design decisions inherent in an algorithm that follows this outline:

1. How should the **initial classifier** be created (step 1)?

2. What is the **inconsistency measure** of a training set (step 2)?

3. What **exit conditions** should be used (step 3)?

4. How is a new **classifier constructed** to reduce inconsistency (step 4)?

We are testing the effect of different answers to these questions, but we can describe the provisional answers.

1. Initial classifier. A classifier is created with randomly selected prototypes, at least one from each class.

2. Inconsistency measure. The Inconsistency Reduction algorithm measures the inconsistency of a derived training set based on the proportion of training instances of distinct classes that are mapped to the same training instance by a set of component classifiers. The measure currently implemented is described below.

3. Exit conditions. The algorithm halts when either of two user-provided thresholds is breached. (1) The inconsistency measure is below a threshold value input by the user. (2) The maximal number of classifiers have been added. The default inconsistency measure threshold is more or less arbitrarily set at 0.2; the default maximal number of classifiers is three.

4. Classifier construction. Classifiers are constructed by incorporating prototypes from distinct classes that have the same derived features. There are two subproblems: (1) select a derived instance that is maximally inconsistent according to some measure, and then (2) through prototype selection, construct a classifier that will reduce the inconsistency of that derived instance.

As to the first problem, the current algorithm chooses a derived instance by determining which instance has the greatest product of the number of instances of distinct classes that are mapped to it. For example, if three instances of class $+$ and two instances of class $-$ are mapped to the derived instance $<--\;>$ by the component classifiers, then $<--\;>$ is assigned the value six for the purpose of selecting the (maximally) inconsistent derived instance.

As to the second problem, the current approach is to select two prototypes from the original training set that are mapped to the derived training instance of greatest inconsistency. Several techniques have been tried to make this selection from among the instances that have inconsistent derived representations, including (1) randomly selecting the two instances from different classes that have inconsistent derived representations; (2) selecting two instances of different classes that are the *closest* of all pairs of instances from different classes whose derived representations are inconsistent; (3) selecting two instances of different classes whose k-nearest neighbors are all of the same class as each of the two instances; and (4) selecting not the instances themselves that have inconsistent representations, but the nearest neighbor of each of those instances. The current approach is (2), but research is on-going.

Once two prototypes are selected from the inconsistent region, any additional prototypes are selected from the classes not represented in the inconsistent region or from the default class if there are only two classes.

Our provisional inconsistency measure can be described formally. First, define a projection operator that takes an instance of the derived training set T_d (which consists of a set of vectors of classifier predictions and the actual class assignment) into its feature vector without the class assignment: $\pi([C_1(x), \ldots C_n(x), c(x)]) = [C_1(x), \ldots C_n(x)]$, where each $C_i(x)$ is the prediction of classifier C_i for the class of instance x and $c(x)$ is the actual class of x.

Let $f : T_o \rightarrow \pi[T_d]$ be the function that takes each original instance $x \in T_o$

and maps each one into a derived instance in the derived training set $\pi[T_d]$ via the component classifiers.

Then $f^{-1} : \pi[T_d] \rightarrow 2^{T_o}$ is the inverse relation that maps a derived instance into the original instances that are mapped by f to that derived instance.

Define $f_C^{-1}(x_d) = \{x|f(x) = x_d \wedge c(x) = C\}$. $f_C^{-1}(x_d)$ represents the set of original instances from class C that are mapped by the component classifiers into the derived training instance x_d.

Let $C_{x_d}^{max} = \arg\max_C |f_C^{-1}(x_d)|$. $C_{x_d}^{max}$ is the class of the greatest number of instances in the region containing x_d.

Then the inconsistency of the derived instance can be defined by

$$a(x_d) = \sum_{C_j \neq C_{x_d}^{max}} |f_{C_j}^{-1}(x_d)|$$

. $a(x_d)$ represents the number of instances that are mapped to x_d except those in the class whose instances are mapped most frequently to x_d.

Finally, the normalized inconsistency of the derived training set T_d can be defined by[3]

$$A(T_d) = [2 \sum_{x_d \in T_d} a(x_d)]/|T_o|$$

.

The idea may be harder to express formally than it really is, and an example may clarify this computation. Suppose that there are two classes, + and $-$, and that we apply two component classifiers to a training set $T_o = \{x_1, \ldots, x_7\}$. Suppose there are five original instances $< x_1, + >, < x_2, + >, < x_3, + >, < x_4, - >, < x_5, - >$ that are mapped into the derived feature vector $< +, - >$. So $C_{<+,->}^{max} = +$, since more instances of class + are mapped to $< +, - >$ than any other class. The inconsistency of $< +, - >$, $a(< +, - >)$, is 2, reflecting the fact that the two instances x_4 and x_5

[3]The factor 2 is not crucial, but reflects an intuition that inconsistent instances come in pairs in a two-class problem. Subjectively, it takes (at least) two instances to have an inconsistency and both instances can be considered inconsistent.

are inconsistent in the derived representation. Suppose that $< x_6, + >$ and $< x_7, + >$ are the only instances mapped to derived instance $< +, + >$. The derived training instance $< +, + >$ has inconsistency 0, since it is not inconsistent: no instances of class $-$ are mapped by the component classifiers to $< +, + >$. The normalized inconsistency of the derived training set is $2 \cdot (2 + 0)/7 = 4/7$, reflecting that four of the seven training instances are inconsistent.

While recursive partitioning algorithms have applied the analogous idea of the impurity of a subset of the instance space, we wish to distinguish work based on that idea from the Inconsistency Reduction algorithm. The impurity of a node in a decision tree, which represents a subspace of the instance space, is at a minimum if it contains only instances from a single class. If the same number of elements of all classes is contained in a subspace, that set is maximally impure [Breiman *et al.*, 1984]. A similar idea is applied here, but the Inconsistency Reduction algorithm is distinguished from algorithms based on impurity in several ways.

1. The Inconsistency Reduction algorithm is not a recursive partitioning algorithm, since the classifier constructed at each iteration in the algorithm has global scope. It is not applied only to the subspace of instances that are deemed inconsistent. This approach is consistent with the philosophy of this research, that giving classifiers global scope may provide predictive information useful to a combining classifier.

2. Recursive partitioning algorithms have handled inconsistent data differently from the Inconsistency Reduction algorithm. Decision tree algorithms such as ID3 often have dealt with inconsistent instances by outputing an "inconsistent classes" value at leaves containing such inconsistent instances or by taking a vote of the instances within such a leaf.

3. The Inconsistency Reduction algorithm applies a different measure of inconsis-

tency, a simple one based on frequency counts, rather than measures used in decision tree algorithms, such as ID3's information gain [Quinlan, 1986].

4. The end to which the Inconsistency Reduction algorithm is applied is different from recursive partitioning algorithms. The Inconsistency Reduction algorithm to applied to effect a re-representation of the original instance space to create a new instance space.

We next describe another approach to selecting complementary component classifiers, based on an analysis of the errors made by each classifier.

5.3.3 *Classifier Selection through Error Orthogonality*

In this section we argue that an analysis of the errors that are made by each component classifier can provide a search bias that can lead to component classifiers that have generalization accuracy for the tested data sets superior to a baseline nearest neighbor classifier. The motivation to examine the sets of errors made by a classifier stems from the observation that a set of component classifiers that all make the same errors do not provide any additional information to a combining classifier that would enable it to boost classification accuracy. We may think of two classifiers that make different errors on a data set as displaying *error orthogonality.* If two classifiers make the same errors, they are not orthogonal in this sense.

Enlisting the term *orthogonality* can be justified by formalizing some of these ideas. Suppose we have two classifiers C_1 and C_2, and a data set $T = \{x_0, x_1, \ldots, x_n\}$. Given a fixed presentation ordering of T we can define a binary error vector E^j that captures some of the classification behavior of a classifier C_j when applied to T. Define

$E_i^j = 0$ if C_j classifies x_i correctly and

$E_i^j = 1$ if C_j classifies x_i incorrectly;

$i = 0, \ldots, n; j = 1, 2$

In general, two real vectors are called *orthogonal* if their dot product is 0. In our context, if two error vectors have a non-zero dot product, then they are not orthogonal. Two classifiers C_1 and C_2 whose error vectors have a non-zero dot product make some of the same errors: $E^1 \cdot E^2 > 0$. $E^1 \cdot E^2$ is the number of instances on which C_1 and C_2 both make misclassifications. If $E^1 \cdot E^2 = 0$, then either C_1 and C_2 both classify each instance correctly, or one classifies it correctly while the other classifies it incorrectly. If a data set T is fixed, we may say that two *classifiers* are orthogonal if their error vectors (defined as above) are orthogonal.

There are many ways that an analysis of the errors made by classifiers might give rise to a bias for selecting classifiers. In the algorithm we present next, the Error Orthogonality algorithm, a search is made for classifiers whose errors on the training set are disjoint. The current algorithm is provisional and we shall propose work to follow this general theme in the investigation of other algorithms.

The current implementation of the Error Orthogonality algorithm has the following broad outline. The idea is to create a pool of classifiers, upon which one can draw, to extract component classifiers that minimize the errors in common according to some measure. In broad outline, the algorithm works as follows:

1. Sample s sets of p prototypes from the training set T. Construct s classifiers using these sampled prototype sets.

2. Based on heuristics, select a set of n classifiers that minimize a measure of error overlap.

This algorithm uses generate-and-test. While a constructive algorithm would be preferable, sampling classifiers is a viable method where the classifiers are small and therefore easily tested, and where sampling gives enough sufficiently diverse classifiers.

Several questions arise in an algorithm that follows this outline.

1. How are the **parameters** set (s, p, n)?

2. What measures of **error overlap** can be used?

3. What **search heuristics** are used to minimize the measure of error overlap, without exhaustive search?

Different answers to these questions give rise to a range of algorithms. We discuss the answers to these questions for the current implementation of the Error Orthogonality algorithm.

1. Parameter settings. The current algorithm does not provide a principled answer to this question. No current algorithms can answer these questions, either, even in a more limited context, such as the number of prototypes that achieves maximal generalization accuracy for a single classifier on a given data set. In practice, $s = 10$ sets of $p = max\{3$,number-of-exposed-classes$\}$ are chosen to find $n = 3$ component classifiers.

2. Measures of error overlap. The current implementation has the goal of minimizing only the number of instances that are in the error sets of all the component classifiers. Other approaches are clearly possible, especially those that use a weighting coefficient to reflect that instances may appear in a number of error sets without appearing in all of them.

3. Search heuristics. In the search for component classifiers with disjoint error sets, we discard classifiers whose error sets are a (proper) superset of others in the pool. A classifier that makes a superset of errors of another can only increase the cardinality of the intersection with a third classifier's errors. The search also is biased in favor of classifiers that make a small number of errors, since, intuitively, it may be easier to find classifiers whose error sets do not intersect such a small set.

At this point we have presented four algorithms for component classifier selection. In the next section we provide results of preliminary experiments testing the generalization accuracy of composite classifiers using components constructed by these four algorithms, for each of three combining algorithms.

5.3.3.1 Experiment

In this section we give preliminary evidence for the hypothesis that algorithms based on sampling, derived training set inconsistency reduction, and error orthogonality yield component classifiers that display generalization accuracy superior to a baseline nearest neighbor algorithm. On three data sets selected from the UCI machine learning accuracy, we tested each of the four component classifier selection algorithms, using each of three combining algorithms. In Tables 5.2, 5.3 and 5.4 for each experiment we give the average generalization accuracy using 10-fold cross validation. Each algorithm was applied to constructing three component classifiers. The results reported for the Random algorithm represent the average taken over 10 samples. For the Best of Random algorithm, the best three of 10 classifiers were used. The number of samples to be selected is a user-specified parameter, and 10 samples is an admittedly small number. However, the sensitivity of these algorithms to the number of samples taken has yet to be determined. (Recall that the classification accuracy of the MC1 sampling algorithm was surprisingly inelastic with respect to the number of samples taken, as shown in Table 4.3.) One point we wish to make is that a very small number of samples of prototypes suffice to give accuracy higher than a nearest neighbor algorithm that incorporates all instances as prototypes.

The algorithms are compared with a baseline nearest neighbor algorithm that uses all training instances as prototypes and applies a Manhattan distance metric, under the column heading "NN". This nearest neighbor baseline accuracy differs from the accuracies reported in Chapter 4 because five-fold cross validation was used in Chapter 4 and in this chapter we have used 10-fold cross validation. Statistical significance was computed using a paired two-sample t-test.

These results provide a starting point for future research on stacked classifier construction. Using a 1-nearest neighbor algorithm as a baseline, these results show generally show the comparable or superior performance of stacked classifiers that

Table 5.2. Component classifier selection algorithms classification percent accuracy, using ten-fold cross validation and ID3 as the combination algorithm. The symbol "†" denotes statistically significant improvement over the baseline 1-nearest neighbor algorithm (NN) at the 0.1 confidence level; "*", significance at the 0.05 confidence level.

Data Set	**NN**	**Random**	**Best of Random**	**Inconsistency**	**Error Orthog.**
Iris	91.3	86.8*	88.7	96.0*	93.3
Breast Cancer	68.9	70.7	72.9	71.8	72.1
Heart Disease	76.0	75.5	80.3*	74.7	79.3†

Table 5.3. Component classifier selection algorithms classification percent accuracy, using ten-fold cross validation and a nearest neighbor algorithm as the combination algorithm.

Data Set	**NN**	**Random**	**Best of Random**	**Inconsistency**	**Error Orthog.**
Iris	91.3	86.2*	88.0	96.0*	92.7
Breast Cancer	68.9	70.7	72.1	71.8	71.4
Heart Disease	76.0	75.3	80.3*	74.0	78.7†

uses ID3 or a k-nearest neighbor combining algorithm and incorporates only several prototypes in a small number of component nearest neighbor classifiers.

However, there is much room for improvement when the baseline is "Best of Random," which takes as components those classifiers that have the highest training set classification accuracy. Only on the Iris data did the Inconsistency Reduction and Error Orthogonality algorithms outperform the Best of Random algorithm. In future work the stacked classifiers should also be compared with the classification accuracy of the (single) component classifier with the highest accuracy on the training set, in order to show that a boost in performance has been achieved. In the next chapter we propose to improve these results or to explain why component classifier selection

Table 5.4. Component classifier selection algorithms classification percent accuracy, using ten-fold cross validation and a voting algorithm as the combination algorithm.

Data Set	**NN**	**Random**	**Best of Random**	**Inconsistency**	**Error Orthog.**
Iris	91.3	82.4*	90.7	87.3	82.7*
Breast Cancer	68.9	56.8*	71.8	70.0	53.2*
Heart Disease	76.0	66.3*	79.0†	66.3*	69.3*

algorithms based on the general ideas of reducing derived training set inconsistency and increasing error orthogonality may not work consistently upon commonly used data sets.

We can make several observations about the performance of these algorithms on these data sets.

Relative Performance of Combining Classifiers. ID3 gave the best nominal performance, and voting was the worst where the component classifiers may display low accuracy, as can happen with the Random, Inconsistency Reduction, and Error Orthogonality algorithms. The performance of ID3 and a nearest neighbor algorithm were very similar as combiners.

Relative Performance of Component Classifier Algorithms. Random selection generally performed the worst of the four algorithms. With the ID3 and nearest neighbor combining algorithms the other three algorithms performed comparably. Component selection using Best of Random outperformed the other algorithms with a voting combiner.

But, on the Iris and Breast Cancer data, selecting the available classifiers with the highest training set accuracy (Best of Random) does not significantly outperform algorithms that select classifiers according to other criteria. Choosing classifiers with lower accuracy on average performed statistically as well as selecting a combination of the best classifiers.

Performance of Composite Classifiers. We have shown that a composite classifier that incorporates a small number of nearest neighbor classifiers, each of which incorporates only a few prototypes, can perform better than a nearest neighbor algorithm that uses all instances as prototypes. Our goal in these initial experiments was not to optimize generalization accuracy by varying the number of component classifiers or the number of prototypes that they apply. We expect that the generalization accuracy of the composite classifiers will be improved as component selection algorithms are explored and refined.

5.4 Integrated Search for Combining and Component Classifiers

Problem 3 from Chapter 1 puts the first two problems together: construct a composite classifier, where both the component and combining classifiers may be varied. This problem appears harder than the previous two subproblems because of the added complication that there may be interactions between the set of component classifiers and the combining classifier. Design of multilayer artificial neural networks also presents this problem of dealing with the interactions between layers. One set of component classifiers may perform best with one combining classifier, and another set of components may perform best with a different combining algorithm.

Designing a composite classifier that minimizes error requires facing the problem of blame and credit assignment. If the composite classifier misclassifies an instance, is the combining classifier the "cause" of the misclassification, or are the component classifiers "to blame?" If the component classifiers are to blame, how is the blame to be apportioned among these components? Finally, once the blame is placed, how can a classifier be reconfigured to eliminate its share of the error? In this section we catalog several approaches to answering these questions, but leave to future work specific algorithms or results to help answer them. The general approaches we mention here are:

1. Cross validation.

2. Local search algorithms with a stochastic component, such as a genetic algorithm or random mutation hill climbing.

3. Heuristics.

4. Stacking of *combining* classifiers.

We discuss each approach in turn.

Cross validation. In an application where the choice of combining classifier is to be made from only a few candidates and only a few sets of component classifiers are considered, one classical approach is to use cross validation of the data set to select a composite architecture. For a given set of component classifiers, this winner-take-all scheme selects the combining classifier with the highest average generalization accuracy across all the plies of a cross-validation partition of the data. One pervasive drawback with using cross validation is that it is computationally intensive. A second is that no guidance is given as to which instantiations of the composite architecture to compare.

Genetic algorithms, random mutation hill climbing. A second approach that is still computationally intensive but does provide some guidance on the classifiers to select is local search with a stochastic component, such as a genetic algorithm (GA) or random mutation hill climbing. In the GA approach, each proposed instantiation of an architecture would be encoded as a chromosome and a fitness function could be defined as the training set classification accuracy (or generalization accuracy on some withheld portion of the training set.) Suitable crossover and mutation operators would have to be defined, along with other parameters and GA architecture components. An appropriate crossover operator would take (the representations of) two composite classifiers and create (the representation of) a third composite classifier, preferably one that is likely to show improved generalization accuracy over its parent classifiers. To apply this approach, the GA would be run until defined termination criteria were satisfied and a composite classifier would be selected from the final population.

Heuristics. Another approach is similar to that taken by Brodley, which is to determine heuristically on the basis of classifier performance on a data set, which combining and component classifiers are likely to yield a classifier of superior accuracy [Brodley, 1992]. Another approach would be to try to extract features of the derived training set that call for the application of one or another combining classifier. This last approach would be difficult and speculative, since it probably involves making progress on the unsolved general problem of determining the best classifier to apply on the basis (of some set of features yet to be determined) of a given set of data.

Stacking of Combining Classifiers. The three approaches just discussed select a single combining algorithm from among the candidates. On the other hand, the philosophy behind this proposal is that it is often useful to combine predictions. Therefore, a three-layer architecture may provide a solution, where a set of component classifiers forms the first layer, the (three) combining classifiers form the second layer, and a single, master combining classifier forms the third layer. We do not expect to adopt this solution, since this architecture only postpones the decision of the top-level combining algorithm, however. On the other hand, the properties of the data derived from the outputs of the second layer combining classifiers may be different from that of the previous layer, in a way that makes the choice of a master combining algorithm clearer. For example, if the data are highly compressed by the time they get to the master level, the choice of master classifier may not matter much.

There are a number of possible approaches to the integrated problem of composite classifier construction. We leave to proposed work the selection of one of them or the adoption of a different tack.

5.5 Conclusion

In this chapter we have presented several approaches to creating composite classifiers. In Chapter 1 we identified three subproblems of this general task: selecting a combining algorithm (Problem 1), searching for component classifiers (Problem 2), and performing these two tasks simultaneously (Problem 3). As to the first problem, we have shown that two combining algorithms (ID3 and nearest neighbor) can combine the predictions of some sets of nearest neighbor component classifiers more accurately than voting. On the second problem, we have suggested four algorithms for searching for component classifiers. Two are immediate outgrowths of random sampling of prototypes. Two of the strategies introduce criteria in addition to in-sample classification accuracy that ought to be considered in the selection of component classifiers: (1) reduction in the inconsistency of the derived training set, and (2) the presence of orthogonal errors on the training set. Finally, on the third problem, we discussed some potential solution paths.

In the final chapter, we discuss the work to complete the dissertation research.

Chapter 6

Proposed Work

6.1 Chapter Organization

The primary objectives of this research are to propose elements of a theory of how to select classifiers for combination and to study composite classifiers that combine the predictions of a set of component nearest neighbor classifiers. In this chapter we propose a strategy for achieving these objectives.

6.2 Possible Research Directions

There are several directions in which this line of research might be continued to complete the dissertation. Work will continue along several of these directions simultaneously. Since we have tested our algorithms only on a small number of data sets, much more empirical work needs to be done. That experimental work is discussed separately in the following section.

Improvement of algorithms for complementary component classifier construction. The preliminary work discussed in the previous chapters was done to show (1) the general utility of classifiers that apply only a small number of prototypes, and (2) the relative utility of four broad strategies to select component classifiers: random sampling, selecting the most accurate, decreasing the inconsistency of the derived training set, and analyzing the errors made by each component classifier. The algorithms we present are our first, tentative steps to capture the intuition behind the broad strategies. The proposed algorithms can be modified to increase their effectiveness and efficiency, and we anticipate proposing other algorithms to select component classifiers. In particular, we shall investigate constructive strategies that search for component classifiers without relying on repeated sampling of prototype sets.

Description of an emerging theory. Our ultimate research goal is to give a theory of how to construct classifiers for stacking. While a comprehensive theory of classifier stacking is beyond the scope of this thesis, the sundry pieces that we identify should be glued into a coherent framework.

Searching the space of nearest neighbor classifiers. In Section 3.7 we presented a model of a k-nearest neighbor classifier with four parameters (the prototypes, k, the distance function, and the class prediction combining function for a set of neighbors). In this proposal we have limited our attention to varying the set of prototypes to construct complementary classifiers. In particular, we have assumed that the same number of prototypes is present for each class, and we should loosen this assumption in future work.

The class prediction combining function, which takes the predictions of a set of neighbors to an instance and outputs a final class prediction, has received little attention in nearest neighbor classifier research. Investigating the utility of combining component classifiers that incorporate different such prediction combining functions may be a useful line of research, and we would propose a brief foray into this somewhat neglected area.

We can also represent a nearest neighbor algorithm as an instantiation of a "local" learning algorithm [Bottou and Vapnik, 1992] captured in 3-tuple: $< T, f, \theta >$ where T is a set of instances, a subset of some universe of instances I, and C is a set of class labels.

$f : I \rightarrow 2^T$ is a function that produces a (local) neighborhood, a subset of T, of an instance in the instance space I.

$\theta : f[I] \rightarrow C$ is a function that takes a neighborhood of instances and returns a class prediction based on that neighborhood.

This more abstract representation suggests that we might have even more flexibility to design nearest neighbor classifiers, especially those that are complementary to a given one. For example, one possibility is to alter the neighborhood-generating function f. We have done preliminary experiments with neighborhoods that are based on walks from an instance, in which an instance-to-instance path is traced for some number of steps. The instances visited in the walk then constitute the neighborhood. Walks in different directions in the instance space could yield neighborhoods that support complementary classifiers.

Identification of characteristic instances. We have begun to explore the notion of *characteristic* instances, instances that are touchstones to the accuracy of classifiers on an entire data set. Our intuition is that by testing a classifier on a small subset of carefully selected instances, we may be able to tell without exhaustive testing the approximate degree of accuracy of the classifier when applied to the entire set of instances. By limiting the testing to a small set of characteristic instances, we shall attempt to accelerate a generate-and-test algorithm for configuring component classifiers. A successful implementation of this approach may be applied to the testing of parameter settings and other components normally done by cross validation, bootstrapping or other computationally intensive techniques.

For example, our proposed approach to the identification of characteristic instances is based on maintaining statistics when a set of random classifiers (of the model class of interest) is each applied to a data set. In our initial implementation, we keep track of the following statistics for each instance in a training set: (1) how many times the instance was correctly classified by a random classifier, (2) the mean accuracy on the whole data set of the classifiers that classified the instance correctly, and (3) the mean accuracy on the whole data set of the classifiers that classified the instance incorrectly. We are exploring ways to identify the set of characteristic instances based on these statistics. For example, if an instance has high mean accuracy for those classifiers that classify it correctly, and low mean accuracy for those classifiers that classify it incorrectly, then that instance may be a good characteristic instance to test. If an instance is almost always correct, if may be a bad choice for a characteristic instance, unless the user wants to include a small number of such instances for a "sanity check."

Next, to test the accuracy of a new classifier that has been generated, testing only would have to be performed on the set of characteristic instances. It is possible that a set of characteristic instances may not be a static set, fixed in advance, but they also may be identified dynamically as tests on characteristic instances are performed to attempt bound the accuracy of a classifier within some confidence interval.

Alternatively, if a suitable algorithm can be found to identify characteristic instances, it should also be determined whether such instances should be used as prototypes, rather than (merely) used for testing the suitability of other instances as prototypes.

6.3 Evaluation

In order to demonstrate empirically the general applicability of the methods we present, they should be tested on a variety of data with different characteristics. Since data sets from the U.C.I. Machine Learning Repository are recognized benchmarks in this area of research [Murphy and Aha, 1994], we propose selecting a fixed collection of data sets that repository for experimental use. The data sets selected should exhibit varying characteristics, such as the number of features and classes, the amount of noise in the features and/or class labels, the attribute value type, and so forth.

We also propose a second, complementary approach to evaluation that requires the creation of artificial data sets. Once we have hypotheses about the characteristics of the data that affect the performance of a composite classifier, we can program a data set generator. This generator would permit us to control the characteristics of data sets in order to show more directly how the performance of a stacked nearest neighbor classifier varies with particular features of a data set. For example, we have already shown that on several data sets the degree of clustering of the data to be classified affects the accuracy of nearest neighbor classifiers that incorporate only a few prototypes (Section 4.7). Thus the degree of clustering (as measured by the Calinski-Harabasz Index or some other measure) would be one degree of freedom that could be varied by such a data generator.

The expected result of our experiments is to show that the techniques we shall explore will lead to small nearest neighbor classifiers that display classification accuracies higher than a nearest neighbor classifier that applies all available instances as prototypes and accuracy higher than the component classifiers. This result will provide concrete evidence for the hypothesis presented in Chapter 1.

6.4 Conclusion

A successful implementation of the ideas presented in this proposal will affect artificial intelligence and machine learning research in several ways. First, we expect this research to show that it is possible to build accurate classifiers from very simple components in many commonly studied domains. Second, it should demonstrate the utility of random sampling and stochastic search techniques for constructing component classifiers in those domains. Third, this research emphasizes the perspective that there are many degrees of freedom inherent in some classifiers and that these can exploited in order to create component classifiers that are complementary.

Appendix A
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 *within cluster sum of squares and cross product matrix* as

$$\sum_{l=1}^{g} \sum_{j=1}^{n_l} (\mathbf{x}_{lj} - \bar{\mathbf{x}}_l)(\mathbf{x}_{lj} - \bar{\mathbf{x}}_l)'$$

Appendix B
Information-gain ratio test

We reproduce here for reference the information-gain ratio of a test for a branching attribute in a decision tree. We follow the description given by Quinlan [Quinlan, 1986] and set out by Brodley [Brodley, 1994, Appendix B, p. 95]. The theory behind the test is given by Quinlan also. We give the two-class test; the multi-class case is a straightforward extension.

Key:
p: number of positive instances in the training set
n: number of negative instances in the training set
v: number of distinct values for attribute A
p_i: number of positive instances in which attribute A has value i
n_i: number of negative instances in which attribute A has value i

Define

- $gain(A) = I(p,n) - E(A)$ (the information gain of testing attribute A) where

- $I(p,n) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$ (information required for classification)

- $E(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(p_i, n_i)$ (the expected information required for a tree with root A)

- $IV(A) = -\sum_{i=1}^{v} \frac{p_i+n_i}{p+n} log_2 \frac{p_i+n_i}{p+n}$ (the information of the values of A)

Appendix C
Description of Cleveland Heart Disease dataset features

The following features are present in the University of California at Irvine Machine Learning Repository Cleveland Heart Disease dataset. Feature descriptions are reproduced verbatim from documentation associated with that dataset.

age: age in years

sex: sex (1 = male; 0 = female)

cp: chest pain type

- Value 1: typical angina
- Value 2: atypical angina
- Value 3: non-anginal pain
- Value 4: asymptomatic

trestbps: resting blood pressure (in mm Hg on admission to the hospital)

chol: serum cholestoral in mg/dl

fbs: (fasting blood sugar $>$ 120 mg/dl) (1 = true; 0 = false)

restecg: resting electrocardiographic results

- Value 0: normal
- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV)
- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria

thalach: maximum heart rate achieved

exang: exercise induced angina (1 = yes; 0 = no)

oldpeak: ST depression induced by exercise relative to rest

slope: the slope of the peak exercise ST segment

– Value 1: upsloping

– Value 2: flat

– Value 3: downsloping

ca: number of major vessels (0-3) colored by flourosopy

thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

num: The presence of heart disease in the patient. It is integer-valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

Bibliography

[Abelson *et al.*, 1985] Abelson, H.; Sussman, G.; and Sussman, J. 1985. *Structure and Interpretation of Computer Programs.* MIT Press, Cambridge, MA.

[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.

[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.

[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.

[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, Dept. of Statistics, University of California, Berkeley, CA.

[Breiman, 1994] Breiman, L. 1994. NIPS*94 Tutorial, Statistics and Nets: Understanding Nonlinear Models from Their Linear Relatives.

[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. Dept. of Computer Science Technical Report 94-61, University of Massachusetts, Amherst, MA.

[Buntine, 1991] Buntine, W. 1991. Classifiers: A Theoretical and Empirical Study. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, San Mateo, CA. Morgan Kaufmann. 638–644.

[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, University of Massachusetts, Amherst, MA.

[Cardie, 1994] Cardie, C. 1994. *Domain-Specific Knowledge Acquisition for Conceptual Sentence Analysis*. Ph.D. Dissertation, Dept. of Computer Science, University of Massachusetts, Amherst, MA.

[Chang, 1974] Chang, C. L. 1974. Finding Prototypes for Nearest Neighbor Classifiers. *IEEE Transactions on Computers* c-23:1179–1184.

[Clement, 1989] Clement, 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, 1991] Dasarathy, B. V. 1991. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA.

[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.

[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.

[Duda and Hart, 1973] Duda, R. O. and Hart, P. E. 1973. *Pattern Classification and Scene Analysis.* John Wiley, New York.

[Edelman, 1993] Edelman, S. 1993. Representation, Similarity, and the Chorus of Prototypes. Technical report, Weizmann Institute of Science, Israel.

[Efron, 1979] Efron, B. 1979. Computers and the Theory of Statistics: Thinking the Unthinkable. *SIAM Review* 21:460–480.

[Fahlman and Lebiere, 1990] Fahlman, S.E. and Lebiere, C. 1990. The Cascade Correlation Architecture. *Advances in Neural Information Processing Systems* 2:524–532.

[Frean, 1990] Frean, M. 1990. The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation* 2:198–209.

[Gates, 1972] Gates, G. W. 1972. The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory* 431–433.

[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.

[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.

[Hart, 1968] Hart, P. E. 1968. The Condensed Nearest Neighbor Rule. *IEEE Transactions on Information Theory (Corresp.)* IT-14:515–516.

[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.

[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.

[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.

[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.

[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.

[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.

[Lakoff, 1987] Lakoff, G. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind.* University of Chicago Press, Chicago.

[Langley, 1993] Langley, P. 1993. Induction of Recursive Bayesian Classifiers. In *Proceedings of the 1993 European Conference on Machine Learning*, Berlin. Springer Verlag. 153–164.

[Lesser *et al.*, 1994] Lesser, V. R.; Nawab, S. H.; and Klassner, F. I. 1994. IPUS: An Architecture for the Integrated Processing and Understanding of Signals. *Submitted to Artificial Intelligence.*

[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.

[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.

[Mitchell and Holland, 1993] Mitchell, M. and Holland, J. H. 1993. When Will a Genetic Algorithm Outperform Hill-Climbing? Technical report, Santa Fe Institute.

[Moore, 1990] Moore, A. W. 1990. Acquisition of Dynamic Control Knowledge for a Robot Manipulator. In *Proceedings, Seventh International Conference on Machine Learning*, Austin, TX. Morgan Kaufmann. 244–252.

[Murphy and Aha, 1994] Murphy, P. M. and Aha, D. W. 1994. UCI 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.

[Papadimitriou and Steiglitz, 1982] Papadimitriou, C.H. and Steiglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ.

[Perrone, 1993] Perrone, M.P. 1993. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*. Ph.D. Dissertation, Brown University, Providence, Rhode Island.

[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.

[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.

[Reilly *et al.*, 1987] Reilly, D.L.; Scofield, C.; Elbaum, C.; and Cooper, L.N. 1987. Learning System Architectures Composed of Multiple Learning Modules. In *IEEE First International Conference on Neural Networks*, San Diego, CA. IEEE. II495–II503.

[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.*, 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, 1977] Rissland, E. L. 1977. *Epistemology, Representation, Understanding and Interactive Exploration of Mathematical Theories*. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, MA.

[Rissland, 1978] Rissland, E. L. 1978. Understanding Understanding Mathematics. *Cognitive Science* 2:361–383.

[Rissland, 1981] Rissland, E. L. 1981. Constrained Example Generation. Dept. of Computer Science Technical Report 81-24, University of Massachusetts, Amherst, MA.

[Rissland, 1989] Rissland, E. L. 1989. Case-Based Reasoning, Introduction to the Proceedings. In *Proceedings: Case-Based Reasoning Workshop*, Pensacola Beach, FL. Morgan Kaufmann.

[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.

[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. Technical Report 94-38, Computer Science Department, Brown University, Providence, RI. (PhD Thesis).

[Saxena, 1991] Saxena, S. 1991. *Predicting the Effect of Instance Representations on Inductive Learning.* Ph.D. Dissertation, University of Massachusetts, Amherst, MA.

[Schaffer, 1994] Schaffer, C. 1994. Cross-Validation, Stacking and Bi-Level Stacking: Meta-Methods for Classification Learning. In P. Cheeseman, R.W. Oldford, editor, *Selecting Models from Data: Artificial Intelligence and Statistics IV.* Springer Verlag, New York, NY. 51–59.

[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.

[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, 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.

[Smith and Medin, 1981] Smith, E. E. and Medin, D. L. 1981. *Categories and Concepts.* Harvard, Cambridge, MA.

[Sobol', 1974] Sobol', I. M. 1974. *The Monte Carlo Method.* The University of Chicago Press, Chicago, IL.

[Stanfill and Waltz, 1986] Stanfill, C. and Waltz, D. 1986. Toward Memory-Based Reasoning. *Communications of the ACM* 29:1213–1228.

[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.

[Tomek, 1976] Tomek, I. 1976. An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6:448–452.

[Utgoff, 1989] Utgoff, P.E. 1989. Perceptron Trees: A Case Study in Hybrid Concept Representations. *Connection Science* 1:377–391.

[Utgoff, 1995] Utgoff, P.E. 1995. Decision Tree Induction Based on Efficient Tree Restructuring. Technical Report 95-18, Dept. of Computer Science, University of Massachusetts, Amherst, MA.

[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.

[Wettschereck and Dietterich, 1992] Wettschereck, D. and Dietterich, T. 1992. Improving the Performance of Radial Basis Function Networks by Learning Center Locations. In *Advances in Neural Information Processing Systems, 2*, San Mateo, CA. Morgan Kaufmann.

[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.

[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.