AN ABSTRACT OF THE DISSERTATION OF

Dietrich Wettschereck for the degree of Doctor of Philosophy in Computer Science presented on June 7, 1994.

Title: A Study of Distance-Based Machine Learning Algorithms

Abstract approved: Redacted for privacy

Thomas G. Dietterich

Distance-based algorithms are machine learning algorithms that classify queries by computing distances between these queries and a number of internally stored exemplars. Exemplars that are closest to the query have the largest influence on the classification assigned to the query. Two specific distance-based algorithms, the nearest neighbor algorithm and the nearest-hyperrectangle algorithm, are studied in detail.

It is shown that the k-nearest neighbor algorithm (kNN) outperforms the first-nearest neighbor algorithm only under certain conditions. Data sets must contain moderate amounts of noise. Training examples from the different classes must belong to clusters that allow an increase in the value of k without reaching into clusters of other classes. Methods for choosing the value of k for kNN are investigated. It is shown that one-fold cross-validation on a restricted number of values for k suffices for best performance. It is also shown that for best performance the votes of the k-nearest neighbors of a query should be weighted in inverse proportion to their distances from the query.

Principal component analysis is shown to reduce the number of relevant dimensions substantially in several domains. Two methods for learning feature weights for a weighted Euclidean distance metric are proposed. These methods improve the performance of kNN and NN in a variety of domains.

The nearest-hyperrectangle algorithm (NGE) is found to give predictions that are substantially inferior to those given by kNN in a variety of domains. Experiments

performed to understand this inferior performance led to the discovery of several improvements to NGE. Foremost of these is BNGE, a batch algorithm that avoids construction of overlapping hyperrectangles from different classes. Although it is generally superior to NGE, BNGE is still significantly inferior to kNN in a variety of domains. Hence, a hybrid algorithm (KBNGE), that uses BNGE in parts of the input space that can be represented by a single hyperrectangle and kNN otherwise, is introduced.

The primary contributions of this dissertation are (a) several improvements to existing distance-based algorithms, (b) several new distance-based algorithms, and (c) an experimentally supported understanding of the conditions under which various distance-based algorithms are likely to give good performance.

A Study of Distance-Based Machine Learning Algorithms

by

Dietrich Wettschereck

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Completed June 7, 1994

Commencement June 1995

Doctor of Philosophy dissertation of Dietrich Wettschereck presented on June 7, 1994

APPROVED:

Redacted for privacy

Major Professor, representing Computer Science

Redacted for privacy

Chair of Computer Science Department

Redacted for privacy

Dean of Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Redacted for privacy

Dietrich Wettschereck, Author

ACKNOWLEDGEMENT

I am indebted to my adviser Tom Dietterich, who was always available to answer my questions or to point me towards issues I should investigate. In addition, he always gave me the chance to pursue my own research endeavors. Tom, your "bias" for guiding me as a graduate student was most appropriate. I really enjoyed working with you for these five years, thank you.

A number of people have contributed to improve the presentation of my research in this dissertation. Most notably, Tom Dietterich, Prasad Tadepalli, Kathy Astrahantseff, and Bill Langford gave extremely helpful comments during the revisions of this dissertation. Without their help, this dissertation would be substantially more difficult to read and probably impossible to understand.

This research was supported in part by NSF Grant IRI-8657316, NASA Ames Grant NAG 2-630, and gifts from Sun Microsystems and Hewlett-Packard. My final year at Oregon State University was financed in part through scholarships from the Oregon Sports Lottery and from the Ryoichi Sasakawa Young Leaders Fellowship Fund.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

LIST OF FIGURES (CONTINUED)

LIST OF FIGURES (CONTINUED)

LIST OF FIGURES (CONTINUED)

LIST OF TABLES

LIST OF TABLES (CONTINUED)

Chapter 1
Introduction

The primary goal of research in the field of *Machine Learning* is to develop methods that enable machines to accomplish tasks for which no known or efficient algorithm exists. The term "learning" indicates that these methods should modify their behavior in response to given stimuli. These stimuli can be unrefined knowledge in terms of rules or hypotheses, or specific instances of the task at hand. Machine learning has been used to solve a variety of tasks, including, but not limited to, recognition of hand written [Fla94] or spoken text [HCG93], classification of stars [FWD93], steering of a vehicle [Pom93], and prediction of stock market developments [Ref92].

Many of the tasks to which machine learning techniques are applied are tasks that humans can perform quite well. However, humans often cannot tell how they solve these tasks. For example, when listening to speech, people recognize that a certain sequence of sounds represents a certain word. However, the most that human subjects can generally say is: "Well, it sounds like an *'A'*, but I do not know why that is so." Inductive supervised learning is able to exploit the human ability to assign labels to given instances without requiring humans to explicitly formulate rules that do the same. These labeled instances are then analyzed by inductive supervised learning algorithms to learn specific tasks.[1] Supervised learning is formally defined as the task of learning from a set of training examples. These training examples are represented by input vectors and their corresponding classifications (labels, classes, output classes). The goal of supervised learning is to classify input vectors that were not necessarily part of the training set and whose classifications are unknown. The

[1] Another form of machine learning is *speed-up learning* where a complete set of rules defining the task (for example, chess) is known, but an evaluation of these rules to determine, say, the next move in chess, is intractable.

ability of an algorithm to classify such unseen instances is called its *generalization* ability.

Theory of inductive machine learning tells us that there cannot be a universal algorithm that can reliably learn any unknown function from a finite set of examples, because that set could have been generated by any of a large number of functions [Val84, Ang88]. In other words, the problem of finding a function that maps a finite set of possibly noisy input vectors onto corresponding output vectors is ill-posed. Hence, additional assumptions about the nature of the function to be learned must be made. Such assumptions must be built into any learning algorithm to enable it to generalize beyond the given training data. These assumptions are called the *bias* of the algorithm [Mit90, Ren86, GBD92]. Introduction of a bias into an algorithm enables that algorithm to find a unique solution to an ill-posed problem. However, the algorithm will be unable to learn the task if the bias is illsuited. The more specific (restrictive) the bias is, the fewer training examples will be sufficient to build the classifier, but the more likely it will be that the function learned will not be the correct one.

There exists a vast number of biases which can be partitioned into two groups: *preference biases* and *restricted hypothesis space biases.* Algorithms employing a bias from the first group have some notion of nominal order of hypotheses, and try to find the "most preferred" hypothesis which fits the data. Decision tree algorithms, for example, try to find rectangular partitions of the training data using a small number of input features such that each region contains only examples from a single class. That is, decision tree algorithms prefer small (shallow) trees over large ones. Preference biases obtain their justification from principles such as the minimum description length principal [Ris78] and Occam's razor [BEHW87, Hau88]. Algorithms employing a restricted hypothesis space bias assume that decision boundaries[2] have a specific shape. For example, the perceptron algorithm assumes that examples are linearly separable, and multi-layer perceptrons use a pre-specified number of hyperplanes.

[2]Decision boundaries are defined as the boundaries between data points from different classes.

Most learning algorithms that perform well in a variety of applications employ a preference bias (ID3 [Qui86], Cascade-correlation [FL90]). However, best results in specific applications such as speech or text recognition are generally achieved by algorithms employing only restricted hypothesis space biases [LBD+90]. It is also important to note that biases are not exclusive. That is, different "sub-biases" can be combined to obtain a new bias. For example, a sub-bias that is often incorporated into the bias of an algorithm is the Min-feature bias [Alm91], in which hypotheses using a smaller number of features are preferred. Another sub-bias concerns how missing features are dealt with by an algorithm.

In this dissertation we will be particularly interested in the bias employed by distance-based algorithms (DBA). Distance-based algorithms are exemplar[3] based algorithms that compute the output for a given instance exclusively from a combination of internal parameters and the distance between that instance and each exemplar (see Section 2.1.1 for a formal definition of DBAs). Part of the bias of every DBA are the assumptions that exemplars most similar to the query have the largest influence on its classification, that input features are un-correlated, and that all input features should be used to determine the most similar exemplars.[4]

Distance-based algorithms are developed, studied, and refined in a variety of fields (statistics [Che84], machine learning [WD94a], pattern recognition [DH73], vision [Low94], and cognitive psychology [Ros78, SM81]) and are used to solve a large number of tasks [LL90, Lee91, Fog92, SL92, WD92, CS93, SLCD93, Low94]. The earliest example of a DBA is the nearest neighbor classifier [FHJ51]. The most recent additions to the DBA family include certain artificial neural networks [NL91, PG89, BL88].

---

[3] An exemplar is either a specific training example or a generalization of a number of examples. For example, the average of the input vectors of several examples or a rectangle built to cover a number of examples may constitute an exemplar.

[4] Different DBAs employ different methods for determining the weight of each feature during distance computation.

A number of properties of DBAs explain the continuing and growing interest in these algorithms. The foremost reason for this interest is that DBAs have shown excellent performance in a variety of applications [LL90, Lee91, Fog92, SL92, WD92, CS93, SLCD93, Low94]. For example, Wettschereck and Dietterich [WD92] have shown that one DBA, which uses Generalized Radial Basis Functions [PG89], outperforms all other methods, including multi-layer perceptrons [RJ86] and Decision Trees [Qui86], in the task of mapping English text to speech. Lowe [Low94] has obtained similar results with another distance-based algorithm.

A further benefit of DBAs is that exemplars can often be interpreted as prototypical instances of the task learned. For example, exemplars constructed by the BNGE algorithm introduced in Section 5.4.4 can be directly mapped onto human readable rules. These rules can then be used to evaluate the classifier or to increase the willingness of the user to accept the classifier's decision.

An additional advantage of distance-based algorithms is that DBAs generally require very short training phases. For example, training for nearest neighbor simply means storing the data. Radial Basis Function Networks can be constructed incrementally [Pla90, Omo92, CL93, Fri93]. These short training phases result from the fact that distance-based algorithms are local algorithms. They classify each query by considering only the local neighborhood of the query. In many cases it is not necessary to have knowledge of the global structure of a task to make decisions that are accurate. This ignorance towards the global structure of the task may, in some cases, lead to inefficiencies. To this date, however, it has not been conclusively shown that any global algorithm consistently outperforms all local algorithms in any specific task.

Some of the shortcomings of DBAs are deterioration of performance with increasing dimensionality (the so-called "curse of dimensionality", [Bel61, Hub85]) as well as sensitivity to the choice of metric used to compute distances and the feature selection algorithm. There are also computational disadvantages of DBAs such as high memory consumption and slow classification. Much of the research concerning

distance-based algorithms – including the research reported in this dissertation – revolves around these issues.

Well-developed theory exists for some distance-based algorithms. In particular, the first nearest neighbor algorithm and radial basis function networks have been shown to be able to learn any possible function (given a sufficiently large sample) [DH73, PS93].

Distance-based algorithms have a number of advantages that make this family of algorithms interesting. As with any algorithm there are also disadvantages. This study is necessary to better understand how to best utilize these algorithms.

1.1 Objectives of this Dissertation

Despite the fact that distance-based algorithms have been studied in a variety of fields, our understanding of the behavior of specific distance-based algorithms and the generalized behavior of the family is still limited. In this dissertation, we will focus on obtaining a better understanding of (a) the relationship between the first nearest neighbor algorithm (NN) and the k-nearest neighbor algorithm (kNN), (b) how to obtain the best performance when employing kNN, and (c) the relationship between NN, kNN and the nearest hyperrectangle algorithm (NGE). Specifically, the following issues will be addressed:

- When does kNN outperform NN? The kNN algorithm requires estimation of the parameter k. An understanding of when it is necessary to estimate k, and when it is better to prefer the simpler NN algorithm, is important to be able to best apply these algorithms.

- How should the value of k be chosen? As with any algorithm with free parameters, a variety of methods exist for choosing these parameters. In this dissertation, we will set out to determine the most efficient and reliable approach for estimating the value of k.

- Can the distances of the k nearest neighbors of a query be used to estimate the query's label more reliably? The most commonly implemented version of kNN disregards information contained in the distances that the nearest neighbors have to the query. We will research whether this information can always be utilized to the advantage of kNN.

- How should the distance metric be chosen? Choosing the proper distance metric is the most important factor influencing the generalization accuracy of any distance-based algorithm. We will study several algorithms for choosing the proper feature weights for distance-based algorithms.

- Nearest neighbor algorithms minimize the amount of computation that must be conducted at learning time at the cost of increased classification time. The nearest-hyperrectangle algorithm (NGE), on the other hand, combines sets of input examples into hyperrectangles to obtain a more compact representation of the training data. We will investigate how the NGE algorithm compares to NN and kNN, what improvements can be made to the NGE algorithm, and how it can be made competitive with kNN.

1.2 Approach

The algorithms studied in this dissertation will be compared empirically. Empirical studies are necessary due to the fact that it is often extremely difficult or impossible, to obtain any analytical results regarding the behavior of learning algorithms. Many of the results that are obtained from analytical studies are also difficult to interpret with respect to real world problems. Further, an analysis of an algorithm's behavior in a specific domain requires knowledge of the exact characteristics of that domain. That knowledge is generally not available. Empirical studies, on the other hand, can be conducted to explore the abilities and shortcomings of an algorithm without exact knowledge of the underlying distribution of the data. General results can be obtained from empirical studies if a large number of domains is chosen to compare algorithms.

An important difference between this empirical study and most other empirical studies [KBC88, MSTG89, WK89, LL90, NL91, WD92, Hol93] is that similar algorithms within one family are compared. Comparison of similar algorithms highlights dissimilarities that can explain observed performance differences. An explanation of observed performance differences carries substantially more information than merely stating that a significant difference was or was not observed. A few other empirical studies – notably Mingers [Min89], Aha [Aha90], and Dietterich et al. [DHB90] employed an approach similar to the one employed here, thereby significantly improving our understanding of learning algorithms.

1.3 Overview of this Dissertation

This dissertation presents and evaluates a variety of methods aimed at improving our understanding and the performance of nearest neighbor and nearest hyperrectangle algorithms.

In Chapter 2, detailed descriptions of the basic algorithms studied in this dissertation are presented along with descriptions of other representative distance-based algorithms. Descriptions of the evaluation domains are given in Section 2.2. Chapter 2 also summarizes the experimental methods employed throughout this research and introduces procedures not directly related to distance-based algorithms such as principal component analysis, the mutual information procedure, and cross-validation.

Six synthetic data sets are employed in Chapter 3 to determine the conditions under which the k-nearest neighbor algorithm is likely to outperform the first-nearest neighbor algorithm.

Chapter 4 presents a detailed study of the k-nearest neighbor algorithm. It begins by assuming that a single value of k is sufficient to classify all queries, and determines the most efficient way to estimate the optimal value for k. Subsequently, several methods for choosing different values of k for differen queries are proposed and evaluated. The k-nearest neighbor algorithm with simple majority voting is compared to kNN with weighted voting in Section 4.2. In the final part of Chapter 4,

the issue of how to pre-process features and to estimate good feature weights is investigated.

The nearest-hyperrectangle algorithm is studied in detail in Chapter 5 and compared to the k-nearest neighbor algorithm. A hybrid k-nearest neighbor and nearest-hyperrectangle algorithm is introduced in Section 5.6.

A summary of the results obtained from the experiments conducted in this dissertation is given in Chapter 6 followed by recommendations regarding the best use of the k-nearest neighbor and the nearest-hyperrectangle algorithms. Chapter 6 is concluded with an overview of possible extensions to the work reported here.

A summary of the generalization accuracies obtained with the methods studied in this dissertation is given in the appendix for reference so that results reported in this dissertation can be compared to results obtained by other researchers.

Chapter 2
Framework

The specific distance-based algorithms studied in this dissertation and the evaluation domains used are introduced in this chapter. The experimental methods used throughout the thesis are also explained. Finally, algorithms for pre-processing, clustering, and feature weight computation are specified.

2.1 A Review of Distance-Based Algorithms

The definition of distance-based algorithms (DBAs) is relatively broad and includes a variety of algorithms. Selected representatives from each sub-group of DBAs are described in this chapter. This section introduces and defines all of the major distance-based algorithms. At the end of the description of each the algorithm, a formal definition of how each DBA computes its outputs is given. Issues of training are ignored in this chapter. Studied in detail in this dissertation are three of the most basic DBAs: the nearest neighbor algorithm, the weighted vote k-nearest neighbor algorithm, and the nearest hyperrectangle algorithm (NGE). With the exception of the NGE algorithm, all DBAs presented here have one feature in common: The number of their internal parameters is either fixed or is determined via cross-validation. The NGE algorithm incrementally increases the number of exemplars (and therefore the number of its internal parameters) during training.

2.1.1 Distance-Based Algorithms – A Definition

Distance-based algorithms are defined in this dissertation as follows: They are given a set of classified examples (x_i, y_i), $1 \leq i \leq L$. The input x_i of each example is

represented by a vector of real numbers ($x_i \in R^n$). Assume there are C classes, then the output is encoded as an integer ($y_i \in N$), a binary code ($y_i \in \{0,1\}^m$), or a one-out-of-m code ($y_i \in \{0,1\}^m$ with at most one $y_{ij} = 1$). Distance-based algorithms construct a set of $r \leq L$ weighted exemplars (u_i, v_i, w_i), $u_i \in R^n$, $v_i \in N$ or $v_i \in \{0,1\}^m$ and $w_i \in R$ where u_i represents the input of an exemplar (for example, either a specific training example or a hyperrectangle), v_i its classification, and w_i the weight of the exemplar. Given a query $q \in R^n$ and a distance function d, the DBA output for q depends only on q's distance to the exemplars and possibly the output values stored with the exemplars and weights attached to the exemplars. Formally: output$(q) = f(d(q,u_1), d(q,u_2), ..., d(q,u_r), v_1, v_2, ..., v_r, w_1, w_2, ..., w_r)$, for some function f.

2.1.2 The Nearest Neighbor and k-Nearest Neighbor Algorithms

One of the most venerable algorithms in machine learning is the nearest neighbor algorithm (NN,[FHJ51, Seb62, CH67, Cov68, DH73, CS93], see also [Das91] for a survey of the literature). The entire training set is stored in memory. To classify a new example the distance is computed between the example, and each stored training example and the new example is assigned the class of the nearest neighboring example. More generally, the k nearest neighbors are computed, and the new example is assigned the class that is most frequent among these k neighbors (this will be abbreviated as kNN). Ties are broken arbitrarily in favor of the class with the smallest index among the ties. The optimal value of k can be estimated via leave-one-out cross-validation ([WK91] and Section 2.4.8). Ties during cross-validation are broken in favor of smaller ks.

Formally: Given a query q, output(q) = majority $(v_{i_1}, v_{i_2}, ..., v_{i_k})$
with $u_{i_1}, u_{i_2}, ..., u_{i_k}$ the k nearest neighbors of q, $k \geq 1$, $u_j = x_j, v_j = y_j, 1 \leq j \leq L$.

2.1.3 Weighted Vote k-Nearest Neighbor

This algorithm is identical to the k-nearest neighbor algorithm as described in the previous section with one exception. The votes of the k nearest neighbors of a query are given weights inversely proportional to their distances. It will be denoted by kNN_{wv}. Variations of this method have been discussed since Dudani's initial paper [Dud75]. Most recently, Wolpert introduced a weighted vote kNN algorithm which he termed HERBIE [Wol89], and Aha discussed a simple form of the algorithm in his dissertation (IB1 [Aha90]).

All these algorithms compute the output corresponding to a given input from a weighted sum of its k nearest neighbors in the stored set. The weight of each of the k neighbors is inversely proportional to the normalized distance from the instance to be classified.

Formally: Given a query q, $\text{output}(q) = c$ s.t. $\frac{\sum_{i=1}^{k}(v_{k_i}=c)/d(u_{k_i},q)}{\sum_{i=1}^{k}1/d(u_{k_i},q)}$ is maximal, with $k_1, k_2, k_3, ..., k_k$ indices of the k nearest neighbors of q, $u_j = x_j, v_j = y_j, 1 \leq j \leq L$, and $(v_{k_i} = c) = 1$ if the class of v_{k_i} is equal to c and 0 otherwise, $c = 1,...,C$. To avoid division by 0, 1 was added to all distances computed in domains that may contain several identical examples (Led Display and Voting domains).

2.1.4 Nested Generalized Exemplar Theory

Salzberg [Sal91] described a family of learning algorithms based on nested generalized exemplars (NGE). In NGE, an exemplar is a single training example, and a generalized exemplar is an axis-parallel hyperrectangle that may cover several training examples. These hyperrectangles may overlap or nest. The NGE algorithm grows the hyperrectangles incrementally as training examples are processed.

Once the generalized exemplars are learned, a test example can be classified by computing the distance between the example and each of the generalized exemplars. If an example is contained inside a generalized exemplar, the distance to that generalized exemplar is zero; otherwise the distance between the example and the generalized exemplar is its (weighted) Euclidean distance to the closest side of the

exemplar. The class of the nearest generalized exemplar is output as the predicted class of the test example.

Each hyperrectangle[5] H describes a region belonging to exactly one output class and is defined by two exemplars which are located at the upper right hand (u_H^{upper}) and the lower left hand corners (u_H^{lower}), respectively.

An instance is classified by the smallest hyperrectangle it falls into, or by the hyperrectangle it is closest to if it is not contained in any hyperrectangle.
Formally: Given a query q, output(q) = v_H with

$$H = \begin{cases} \text{argmin}_U(area(u_U^{upper}, u_U^{lower})) & \forall\ U : u_{U_j}^{lower} \leq q_j \leq u_{U_j}^{upper}, 1 \leq j \leq n \\ \text{argmin}_U(d(q, u_U^{upper}, u_U^{lower})) & \text{otherwise} \end{cases}$$

where

$$d(q, u_U^{upper}, u_U^{lower}) = w_U \sqrt{\sum_{j=1}^{n} (|q_j - u_{U_j}^{upper}|_+ + |u_{U_j}^{lower} - q_j|_+)^2}$$

and $|x|_+ = 0\ \forall\ x \leq 0$ and $|x|_+ = x$ otherwise.

The following distance-based algorithms are not studied in this dissertation. These algorithms are described here to show the generality of the definition given in Section 2.1.1 and to give the reader an impression of the wide range of algorithms that we consider distance-based.

2.1.5 Variable-Kernel Similarity Metric Learning

Lowe [Low94] suggests placing a Gaussian kernel over each query to determine its output. The k neighbors nearest to the query are determined (Lowe uses $k = 10$). The average distance of the query to its $k/2$ nearest neighbors is used to determine the width of the Gaussian at each query point. Each neighbor's distance from the query is passed through the Gaussian kernel to obtain the weight of that neighbor's vote when the query's class is determined. Lowe uses a gradient descent method to

[5] I have artificially reworded the definition so that it closely matches the descriptions of the other algorithms. Note that two exemplars (H^{upper} and H^{lower}) are involved in the distance computation.

learn the similarity metric (feature weights), and a single, global scaling factor to scale the width of each of the Gaussians.
Formally: Given a query q, output$(q) = c$ s.t. $\frac{\sum_{i=1}^{k} n_i(v_i=c)}{\sum_{i=1}^{k} n_i}$ is maximal with $n_i = \exp(-d(q,u_i)_i^2/2\sigma)$, and $(v_{k_i} = c) = 1$ if the class of v_{k_i} is equal to c and 0 otherwise, $c = 1,...,C$.

2.1.6 Learning Vector Quantization

Kohonen introduced the notion of self-organizing feature maps as a model of the brain's ability to form topology-preserving mappings of sensory inputs. To quote Kohonen, "Visual, somatosensory, etc response signals are obtained in the same topographical order on the cortex in which they were received at the sensory organs." [Koh89].

Learning Vector Quantization (LVQ) algorithms are supervised learning algorithms developed by Kohonen [Koh90b] to improve the classification accuracy of self-organizing maps. The class of an instance is determined by its nearest neighbor (codebook vector) within the feature map. LVQ differs from simple nearest neighbor classification in that supervised learning is employed to find good locations for the codebook vectors. Generally, the number of codebook vectors is chosen to be smaller than the number of training examples. The classification procedure employed by LVQ is identical to that of the basic first-nearest neighbor algorithm.
Formally: Given a query q, output$(q) = v_i$ with u_i the nearest neighbors of q, u_i generally $\neq$ to x_j $\forall j, i$ and $r \ll L$ with r the number of codebook vectors and L the number of training examples.

2.1.7 Radial Basis Functions

The task of learning from examples can be seen as finding an interpolation or approximation function that maps a point in the n-dimensional input space onto a point in the m-dimensional output space. In other words, the learning algorithm must find a function f with: $f : \mathcal{R}^n \rightarrow \mathcal{R}^m$. In the case of interpolation, one basis function

(generally nonlinear) is centered at each training point, and the output is computed as a weighted sum of these basis functions. Broomhead and Lowe [BL88] reviewed the relation between multivariable functional interpolation and adaptive networks. They argued that interpolation often overfits the data and that better generalization could be achieved by using fewer basis functions than there are data points. In that case, the training data is only approximated.

Radially symmetric Gaussian basis function (RBF) networks, as proposed by Moody and Darken [MD88], are the most commonly-used radial basis function networks.

Formally: Given a query q, $\text{output}(q) = \sum_{k=1}^{r} w_k \cdot e^{-\frac{d(q,u_k)^2}{\sigma^2}}$, where r is the number of basis functions and the u_k are either a random subset of the training set or are obtained via application of an unsupervised clustering algorithm (Section 2.4.7) to the training set. Alternative methods for constructing RBF networks have been proposed by many authors ([Pla90, Fri93, CL93, XKO93]).

2.2 Domains

The performance of the algorithms introduced in Section 2.1 will be evaluated in a number of different domains. Most domains chosen for this study have been the object of empirical evaluation by other researchers. This ensures that it will be possible to compare the performance of distance-based algorithms described in this dissertation to other families of algorithms.

The assumption that the results obtained from this study will be of value for other domains is justified for two reasons. First, the domains were chosen without consideration of whether they were well-suited for the algorithms tested (with the exception of the synthetic data sets described below). Second, the relatively large number of domains and the fact that these domains differ in complexity, size, and noise levels should insure the robustness of any conclusions drawn.

The domains considered include two extremely difficult problems, the tasks of mapping English text to speech and of recognizing isolated spoken English letters.

Six data sets were constructed to test specific hypotheses about the behavior of the algorithms studied in a controlled environment. Advantages of artificial domains over "real world" data sets are that the level of noise, number of relevant features, and shape of the decision boundaries of these domains are exactly known before experimentation. Artificial data sets can therefore be used to evaluate the bias of an algorithm and to determine its behavior with respect to increasing levels of noise. Eleven additional domains were obtained from the University of California at Irvine's repository [MA91] of machine learning databases. See [Aha90] for specific details.

In the remainder of this section a short description of each of the domains is given. Characteristic features of the data sets used in this dissertation are also summarized in Table 2.1.

Synthetic Data Sets Six synthetic data sets were constructed so that we would be able to evaluate the different behaviors of the algorithms in domains with properties that were exactly known (Figure 2.1). The *quadrants* and *banded* tasks have axis-parallel decision boundaries, while the *diagonal* task has a diagonal decision boundary. The decision boundary in the *sinusoidal* task is a sine curve, and the *radial* task consists of 5 nested rings of examples from different classes. The *gaussian* task consists of four Gaussian distributions (variance 0.025).

Fisher's Iris Data Set This data set consists of four measurements made by E. Anderson on 150 samples of three species of iris. Fisher [Fis36] used this data set in his classic paper on discriminant analysis. This task is relatively easy, since the species iris setosa is linearly separable from the other two species, and there is very little overlap between the species iris versicolor and iris virginica.

Wine Recognition Data These data are the results of a chemical analysis of wines grown in a single region in Italy, but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three

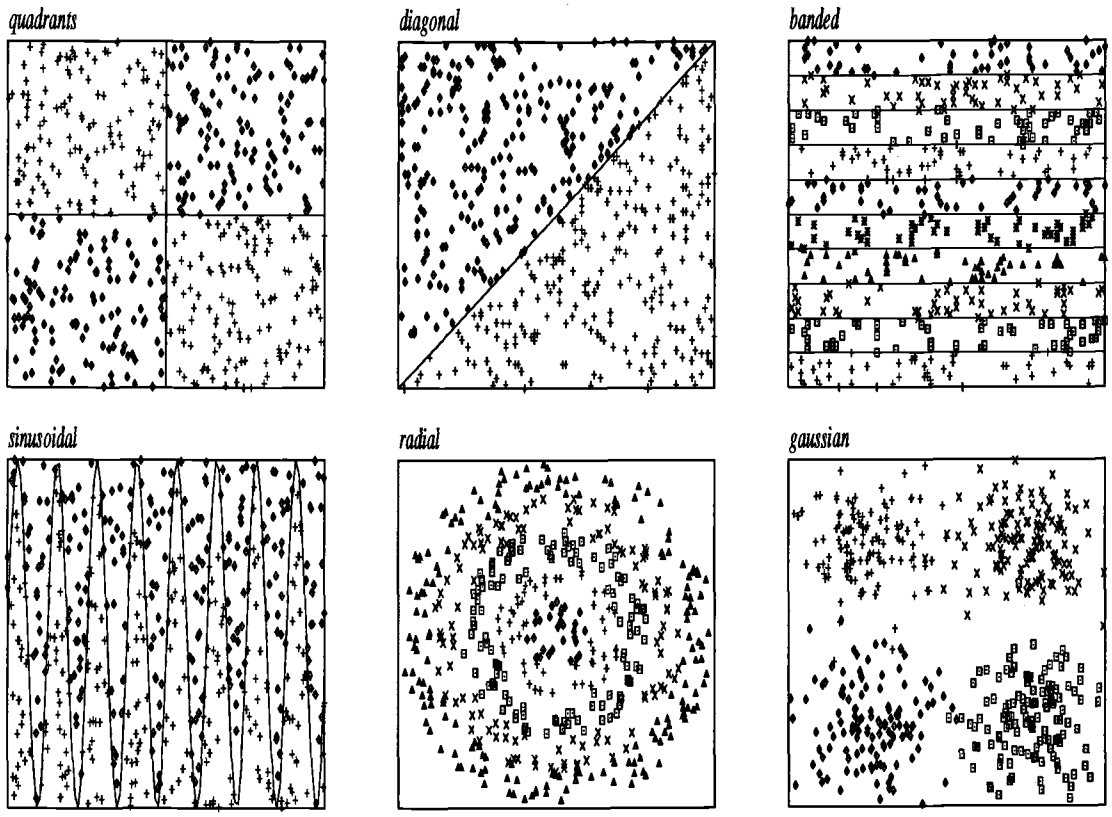


Figure 2.1. The distribution of examples from the different classes in several 2-dimensional artificial data sets. The *quadrants*, *diagonal*, and *sinusoidal* tasks are 2 class problems. The *banded* task has 10 classes, the *radial* task consists of 5 concentric rings and the *gaussian* task contains examples drawn from 4 Gaussian distributions (variance = 0.025). Different symbols indicate different classes. Lines represent the decision boundaries used to label the data.

types of wines. The task is to determine from the constituents the cultivar that was used to grow the wine.

Glass Identification Database The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence—if it is correctly identified! Measurements of 9 chemical compounds extracted from the glass found at the scene of the crime must be used to identify the type of the glass.

Led-7 and Led-24 Display The Led-7 and Led-24 Display domains are artificial domains introduced by Breiman et al [BFOS84]. Both domains have seven boolean inputs indicating whether light-emitting diodes are on or off. Seventeen irrelevant features have been added to Led-7 to create the Led-24 domain. Each attribute value is corrupted with a probability of 0.1. The task is to classify the input as one of the ten digits.

Waveform-21 (40) The Waveform-21 (40) domain is also a noisy artificial domain that was designed by Breiman et al. [BFOS84]. Each instance of this domain contains 21 real-valued attributes (Waveform-40 has an additional 19 irrelevant attributes). Three waves are defined. Each instance is a linear combination of 2 waves, and the classification task is to determine to which of the 3 possible combinations an instance belongs.

Cleveland and Hungarian databases The Cleveland and Hungarian databases (Detrano et al. [DJS+89]) contain cardiological diagnoses. Each input vector describes data obtained from a patent. The task is to determine whether the patient suffers from heart disease. About 20% of the features in the Hungarian database and less than 1% of the features in the Cleveland database are missing.

Congressional Voting database The Congressional Voting database contains little noise; some of the 16 boolean inputs are irrelevant. The target of this application is to determine the political party of United States Congressional representatives from their voting records. This domain is linearly separable, although some input features are missing for some examples.

Isolated Letter Recognition This database contains letters of the English alphabet spoken in isolation. It was recorded at the Oregon Graduate Institute [CMF90].

Table 2.1. Characteristics of domains used as evaluation domains (modified from [Aha90]). B = Boolean, C = Continuous, N = Nominal, S = Symbolic.

Domain	Training Set Size	Test Set Size	Number and Kind of Features	Number of Classes
Quadrants	350	150	2 C	2
Diagonal	350	150	2 C	2
Banded	350	150	2 C	10
Sinusoidal	350	150	2 C	2
Radial	350	150	2 C	5
Gaussian	350	150	2 C	4
Iris	105	45	4 C	3
Wine	125	53	13 C	3
Glass	150	64	9 C	7
Led-7 Display	200	500	7 B	10
Led-24 Display	200	500	24 B	10
Waveform-21	300	100	21 C	3
Waveform-40	300	100	40 C	3
Cleveland	212	91	5 C, 3 B, 5 S	2
Hungarian	206	88	5 C, 3 B, 5 S	2
Voting	305	130	16 B	2
Isolet	1040	1040	617 C	26
Letter Recognition	16000	4000	16 N	26
Letter Recognition$_{BR}$	1000	500	16 N	2
NETtalk	1000	1000	7 S	140

Each input feature encodes the value of a specific time frame of the waveform sample obtained from the speaker. The entire database contains two productions of each letter by 150 speakers. A subset of 40 speakers was used in this dissertation. Cross-validation procedures (Section 2.4.8) have been modified to ensure that cross-validation and testing were conducted on records from distinct speakers.

Frey and Slate's Letter Recognition database This data set was created by D. Slate. Frey and Slate used it to evaluate several *Holland-style adaptive classifier systems* [FS91]. The data set consists of randomly permuted black-and-white rectangular pixel displays of the 26 capital letters in the English alphabet. The original images were obtained from 20 different fonts. Since this data set is rather large (Table 2.1) only the letters **B** and **R** of this domain were used to train and test the classifiers in some experiments. Whenever only these two letters are used, the subscript BR will be appended to the name of the data set. Initial experiments have shown that these two letters are among the two letters that nearest neighbor algorithms confuse most often.

NETtalk The goal of the NETtalk task (Sejnowski & Rosenberg [SR87]) is to learn to pronounce English words by studying a dictionary of correct pronunciations. In this task, each letter to be pronounced is presented to the classifier together with the three preceding and succeeding letters in the word. Output is the phoneme and stress that constitutes the pronunciation of the letter.

2.3 Experimental Methods

To measure the performance of the distance-based algorithms, the training set/test set methodology was employed. Each data set was randomly partitioned into a training set containing approximately 70% of the examples and a test set containing the remaining examples (see also Table 2.1). After training on the training set, the

percentage of correct classifications on the test set was measured. The procedure was repeated a total of 25 times to reduce statistical variation. In each experiment, the algorithms being compared were trained (and tested) on identical data sets to ensure that differences in performance were due entirely to the algorithms. We followed the same procedure to generate learning curves, except that only a subset of the training set was used. The test set along each learning curve was constant, while each larger training set contained all smaller ones.

We have reported the average percentage of correct classifications and its standard error. Two-tailed, paired t-tests were conducted to determine the level of significance at which one algorithm outperformed another. A performance difference was considered significant when the p-value was smaller than 0.05.

Cross-validation (see Section 2.4.8) was performed to estimate optimal settings for free parameters such as k for kNN, M in localKNN (Chapter 4), and the number of seeds for NGE (Chapter 5). Whenever computationally possible, leave-one-out cross-validation [WK91] was employed.

2.4 Other Procedures

2.4.1 Error Correlation Studies

Absolute percentage point differences in classification accuracies on a set of test examples are often not sufficient to determine whether two algorithms share the same strengths and weaknesses. Error correlation studies can often be used to find subtle differences among different methods. If the error correlation turns out to be very low for a set of algorithms, it indicates that a hybrid of these algorithms might be able to overcome some of the errors. A high correlation, however, indicates that the algorithms compared have a very similar bias. In the latter case, other concerns aside from classification accuracy should be used to decide on which algorithm to use. Venn-diagrams [Ven94] can be employed to compare the correlation of up to four methods. Correlation can be expressed in a table when more than four methods are compared.

2.4.2 Preprocessing of Data with a Projection Pursuit Method

We considered applying projection pursuit methods to the data to reduce input dimensionality and decorrelate input features. Since projection pursuit methods compute *linear* projections, they may fail to preserve the information contained in the training data if it contains highly nonlinear structures (Huber [Hub85]). Huber lists the method of principal component analysis as a classical implementation of a specific projection pursuit method. Hence, rather than employing general projection pursuit, we employed only principal component analysis.

2.4.3 Principal Component Analysis

Flury [Flu88] writes:

Principal component analysis can be looked at from three different points of view:

1. *It is a method of transforming correlated variables into uncorrelated ones.*
2. *It is a method for finding linear combinations with relatively large or relatively small variability.*
3. *It is a tool for data reduction.*

The procedure for obtaining the principal components[6] of a $p \times n$ matrix $\mathbf{X}$ of n p-dimensional samples can be summarized as follows: Let $\mathbf{\Psi}$ be the covariance matrix of $\mathbf{X}$. By spectral decomposition, $\mathbf{\Psi}$ can be decomposed into $\mathbf{\Psi} = \beta\mathbf{\Lambda}\beta^T$ where $\beta = (\beta_1, \beta_2, ..., \beta_p)$ is an orthogonal $p \times p$ matrix and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_p)$ is a diagonal matrix of eigenvalues with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$. Then $\mathbf{U} = \beta^T\mathbf{X}$ contains the principal components of $\mathbf{X}$ and has covariance $\text{Cov}(\mathbf{U}) = \text{E}[\mathbf{U}\mathbf{U}^T] = \beta\mathbf{\Psi}\beta^T = \mathbf{\Lambda}$ which indicates that the principal components are pairwise uncorrelated.

Principal component analysis does not assume a specific distribution of the data, such as a normal distribution. However, it will give best results if the data distribution is elliptical but not circular [Mor76]. Principal component analysis can also

[6]This technique is known as the Karhunen-Loéve expansion in the communication theory literature.

be described as a method for finding a new coordinate system such that the sums of the squared distance of each point to its projections on successive axes are minimized [Pea01]. Although the principal components are ordered with respect to their variation in the input data, the variable with the highest predictive relevance may be the variable with the lowest variation in some cases [Ksh72].

The pca-program by A. Weigand from the International Computer Science Institute was used in this dissertation to compute principal components.[7]

2.4.4 Determining Weights by Mutual Information

The purpose of a feature weight mechanism is to give low weight to features that provide no information for classification (e.g., very noisy or irrelevant features), and to give high weight to features that provide reliable information. Hence, a natural quantity to consider is the mutual information [Sha48, McG55] between the values of a feature and the class of the examples. The mutual information between two variables is defined as the reduction in uncertainty concerning the possible values of one variable that is obtained when the value of the other variable is determined [CT91]. In this research, we want to determine the amount of information ("reduction in uncertainty") that knowledge of a feature's value yields in respect to the classification. If a feature provides no information about the class, the mutual information will be zero. If a feature completely determines the class, the mutual information will be proportional to the log of the number of classes (assuming examples from different classes have equal frequency). The mutual information between two discrete random variables is defined as follows:

$$I(X;Y) = H(X) - H(X \mid Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)p(y)}$$

where $H(X)$ is the entropy of a random variable X with probability mass function $p(x)$ and $H(X) = -\sum p(x)\ log_2 p(x)$. In our case, we want to compute the

[7]The program can be obtained via ftp from icsi-ftp.berkeley.edu.

mutual information between class C and feature F_j. The probability distributions of C and F_j are in general unknown, but can be estimated from the training data. Let

- $p(C = c)$ be the probability that the class of any training example equals c.
- $p(F_j = f)$ be the probability that the value of the symbolic or nominal feature F_j of any example equals f.
- $p(C = c \ \wedge \ F_j = f)$ be the joint probability of these two events.
- $nClasses$ be the number of classes.
- nFV be the number of distinct values feature F_j can assume.

Then the mutual information between feature F_j and the classification C is defined as:

$$I(C, F_j) = \sum_{i=1}^{nFV} \sum_{c=1}^{nClasses} p(C = c \ \wedge \ F_j = f_i) \cdot log \frac{p(C = c \ \wedge \ F_j = f_i)}{p(C = c) \cdot p(F_j = f_i)}.$$

For continuous features, we must compute the mutual information between a discrete random variable (the output class) and a continuous random variable (the feature). Assume feature F_j has density $f(x)$ and the joint density of C and F_j is $f(x, y)$. Then the mutual information between C and F_j can be computed as follows:

$$I(C, F_j) = \int_x \sum_{c=1}^{nClasses} f(x, c) \cdot log \frac{f(x, C = c)}{f(x) \cdot p(C = c)} dx$$

A density estimation technique must be employed to estimate the probability distributions $f(x)$ and $f(x, C = c)$ for continuous feature F_j. The kth nearest neighbor density estimate, as described by Silverman [Sil86], was used in this research. This method estimates the density at a certain location from the distance of the kth nearest data point from this location. Formally, the density $f(x)$ at location x is estimated as:

$$\hat{f(x)} = \frac{k - 1}{2N d_k(x)}$$

where N is the sample size and $d_k(x)$ is the distance of the kth nearest neighbor from location x. Initial experiment showed that the value 25 of k gave good results

in most applications. The integral $\int_x$ was numerically approximated by computing the value of $f(x)$ for values of x that varied from 0 to 1 at steps of size 0.002.

The probabilities were estimated from the training data, and missing values were ignored. Note that this feature weight mechanism assumes independence of the features and may therefore lead to inferior performance in domains with highly correlated features.

2.4.5 Distance Metrics

The most commonly used metric for distance computation is Euclidean distance, also called the L^2-norm. This metric was used in all experiments throughout this research unless otherwise noted. Alternative metrics considered were weighted Euclidean distance, where the weights were either learned (Section 4.3.3), computed via the Mutual Information procedure (see also Section 2.4.4), or set to the eigenvalues computed via principal component analysis (Section 2.4.3). The weighted L^p norm between two vectors, q and u, is formally defined as:

$$L^p(q,u) = \left(\sum w_i \times d(q_i,u_i)^p\right)^{\frac{1}{p}}$$

where

$$d(q_i,u_i) = \begin{cases} q_i - u_i & \text{if feature } i \text{ is continuous or nominal} \\ 1 & \text{if feature } i \text{ is symbolic and } q_i \text{ equals } u_i \\ 0 & \text{otherwise} \end{cases}$$

$\frac{1}{p}$ denotes the positive p-th root.

2.4.6 Missing Features

Some of the input features in the Cleveland and Hungarian databases as well as in the Congressional Voting database are missing. It was necessary to modify all algorithms so that they dealt properly with missing features. We adopted Aha's *Ignore* [Aha90, Section 5.2.1] method for handling examples with missing features. This method will ignore any missing features when computing the distance between a given input and

the exemplars. To distinguish missing features from identical features, the distance between two vectors is divided by the square root of the number of features which are known in both vectors at the same time. Formally:

$$d(q,u) = \frac{\sqrt{\sum d(q_i,u_i)}}{\sqrt{\text{number of features } i \text{ for which } q_i \text{ and } u_i \text{ are known}}}$$

where $d(q_i,u_i) = 0$ whenever either q_i or u_i are unknown and $d(q,u) = \infty$ whenever there is not a single feature of q and u known simultaneously.

The reader should note that the performance of any inductive learning algorithm may change substantially in domains where a large number of features is missing, if a different missing-values policy is employed. See [Aha90] for a comparison of three different missing-values policies.

2.4.7 Clustering Algorithms

The K-Means Algorithm The k-means algorithm [Mac67] is one of the most widely used unsupervised clustering algorithms in machine learning. The algorithm finds centers of a user-defined number of clusters. Each cluster center ("mean") is initialized to one random input example. The algorithm iteratively finds the nearest cluster center for each new example and replaces that center by the weighted average of that example and the old center. The weight of the center is given by the number of examples that have been previously averaged into that center.

Rival Penalized Competitive Learning Rival Penalized Competitive Learning (RPCL, [XKO93]) is a straightforward modification of the well known k-means clustering algorithm [Mac67]. Cluster centers in RPCL are initialized outside of the input range covered by the training examples. The algorithm then moves only those cluster centers that are needed into the range of input values, and effectively eliminates the need for cross-validation on the number of clusters in k-means. In this dissertation, a simple version of RPCL with a fixed number of initial clusters

and fixed learning rates was employed. The number of initial clusters was always set to 25, an arbitrarily chosen number that was larger than the expected number of clusters in all cases. A larger number should be used if it is expected that more clusters might be present. The learning rates α_c (rate of movement of nearest cluster center towards training example) and α_r (rate of movement of runner-up cluster center away from training example) were set to 0.05 and 0.002, respectively. These values were employed by Xu et al. [XKO93]. Other values might result in faster convergence and/or a different number of active cluster centers. Hence, if learning time or the number of active clusters are of concern, then these values should be determined experimentally.

2.4.8 Cross-validation

Cross-validation [Sto74, WK91] is a widely used method for estimating optimal values for free parameters. The need for cross-validation stems from the fact that most learning algorithms will *overfit* if all the training data is used to estimate values for free parameters (see also [GBD92]). *Overfitting* means that the learning method will learn overly specific (coincidental) features of the training set. This overly specific "knowledge" will then impair the method's ability to generalize beyond the training data.

The basic idea behind cross-validation is to set aside parts of the training data, train the classifier with different settings of the free parameters on the remaining training data, and then to test all resulting classifiers on the data set aside previously. The setting of free parameters which leads to the best performance on the cross-validation test set is then used to train the final classifier on all training data. One extreme form of cross-validation is leave-one-out cross-validation. Here each training example is set aside in turn, a classifier is constructed for all remaining examples, and then tested on the single example set aside. This method is the computationally most expensive cross-validation method, since N classifiers must be constructed if there are N training examples. On the other hand, the method uses almost all

training data to estimate free parameters, and therefore, should give a very good estimate.

Less computationally expensive cross-validation methods involve setting aside more than one example at a time, which reduces the number of times the classifier must be constructed. In k-fold cross-validation, for example, the training data is divided into k data sets, with each set containing N/k training examples. Leave-one-out cross-validation is equivalent to N-fold cross-validation. It has been recommended [BFOS84] to evenly divide the training examples from the different classes among the k partitions used in k-fold cross-validation when $k > 1$ (this procedure is called stratified cross-validation).

Weiss [Wei91] has shown in experiments with first-nearest neighbor and third-nearest neighbor that two-fold cross-validation may give estimates superior to those obtained with leave-one-out cross-validation for very small samples. The issue of which cross-validation method is most appropriate for kNN is investigated in Section 4.1.1. A computationally efficient alternative to k-fold cross-validation is what will be termed one-fold cross-validation throughout this research. In one-fold cross-validation, one subset of the training data (usually around 20–25% of N) is set aside for cross-validation, and the remaining training examples are used for training. The classifier is trained on that sub-training set and tested on the cross-validation set for different settings of the free parameters until the best setting of the free parameters is found.

Cross-validation may also be used to select a classification method among a set of alternatives [Sch93b]. A computationally more efficient alternative to leave-one-out cross-validation is suggested by Maron & Moore [MM94]. Maron & Moore use statistical means to estimate when one classifier can be believed to be superior to another. When applied to a group of classifiers (for example, kNN with all the different instantiations of k), the classifiers that perform very poorly can be eliminated after a few cross-validation examples have been tested. More cross-validation examples are then tested until either only a single classifier remains or the estimated error rate is believed to be within a certain range of the true error rate.

Chapter 3
An Evaluation of Nearest Neighbor Algorithms

The nearest neighbor algorithm is one of the oldest, most widely used, and successful machine learning algorithms. It has been studied in a variety of environments since Fix and Hodge's initial paper [FHJ51], and innumerable variations of the basic algorithm exist [Das91]. This chapter is devoted to an in-depth study of the nearest neighbor algorithm. Experiments described in this chapter were designed to determine the effect of the number of neighbors that are used to classify a query. It is known that cross-validation on the number of neighbors used to classify a query can lead to improved and/or more robust performance of the nearest neighbor algorithm. However, it is not known unequivocally under which circumstances the k-nearest neighbor algorithm (kNN, k chosen via leave-one-out cross-validation) performs better than the first-nearest neighbor algorithm (NN, $k = 1$).

The k-nearest neighbor algorithm is generally considered to give classification accuracies superior to the first nearest neighbor algorithm. Although a variety of empirical results that support the correctness of that assumption exist [SD86, CU87, JRLW87, Aha90], there are also a few known cases where the first nearest neighbor algorithm performs as well or better than kNN where k is determined via cross-validation [Hol93, SLCD93]. Theoretical results indicate that the first nearest neighbor algorithm can PAC-learn [Val84] any function that can be represented by the union of a finite number of closed hyper-curves of finite size [AKA91]. Cover & Hart [CH67] show that for any size of the sample (training) set there exists no value of $k > 1$ that has a consistently lower error rate than $k = 1$ against all distributions. However, little is known about the causes of the performance differences between NN ($k = 1$) and kNN (k chosen via leave-one-out cross-validation) that are often

observed for specific distributions. Here are two hypotheses that may explain why kNN's performance is often superior to NN's:

H1. Noisy data necessitate large values for k.

H2. The performance of kNN is less sensitive to the choice of distance function used.

The experiments presented below were designed to test these two hypotheses. The six synthetic data sets described in Section 2.2 were used as evaluation tasks.

3.1 H1: Noisy Data.

Hypothesis H1 states that in domains where data points are noisy, larger values of k could be helpful in filtering out some of the noise. Consider, for example, the picture in Figure 3.1. Example b is noisy. The first nearest neighbor algorithm will misclassify point a, because b is its nearest neighbor. With $k = 3$ or $k = 5$, the other neighbors c, d, e and f can "out-vote" b so that a is correctly classified. Distance-based algorithms are susceptible to at least three distinct types of noise: class noise, feature noise, and irrelevant features. Three sets of experiments were conducted to test hypothesis H1, and to determine whether the three kinds of noise had different effects on the nearest neighbor algorithm.

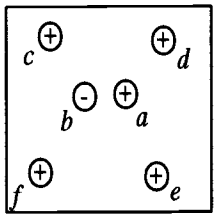


Figure 3.1. An example of a data set with one noisy example. Example b is noisy. First nearest neighbor will misclassify point a, while kNN with $k = 3$ will correcly classify a since d and e "out-vote" b.

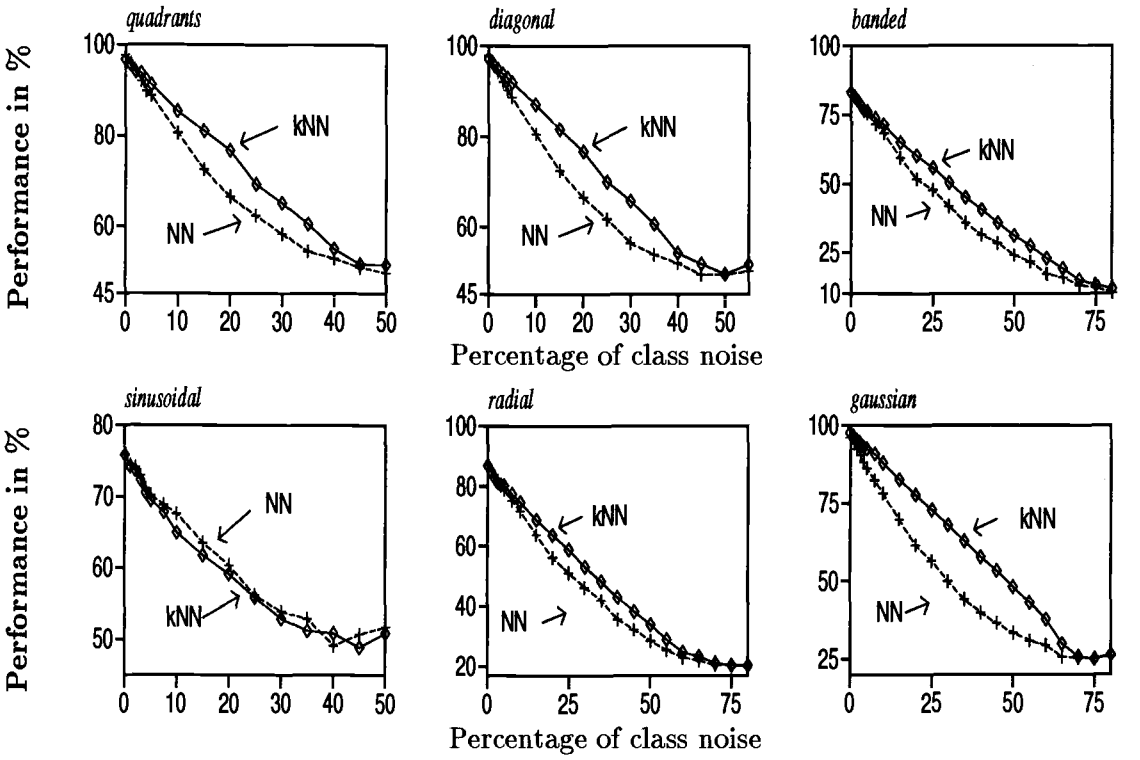


Figure 3.2. The performance of NN and kNN in response to increasing levels of class noise (350 training examples, 150 test examples).

3.1.1 Class Noise

In the first set of experiments, class labels of training (and test) examples were changed with increasing probability. In the noise-free case, leave-one-out cross-validation leads to improved performance only in the *gaussian* task (Figure 3.2, 0% class noise), which is the only task where regions with examples from different classes overlap. In the *quadrants* and *diagonal* tasks, the performance of kNN is even inferior ($p < 0.05$, Table A.1) to NN's. This indicates that the performance of kNN might be improved if the choice of k is biased more strongly toward small values for k (see also Section 4.1.1). For moderate amounts of class noise, cross-validation on k leads to significantly superior performance in all tasks except the *sinusoidal* task.

Why such different effects can be observed in different tasks when class noise is added can be explained by the following argument: If all examples in one class are well enough separated from all examples of other classes, then k could be increased

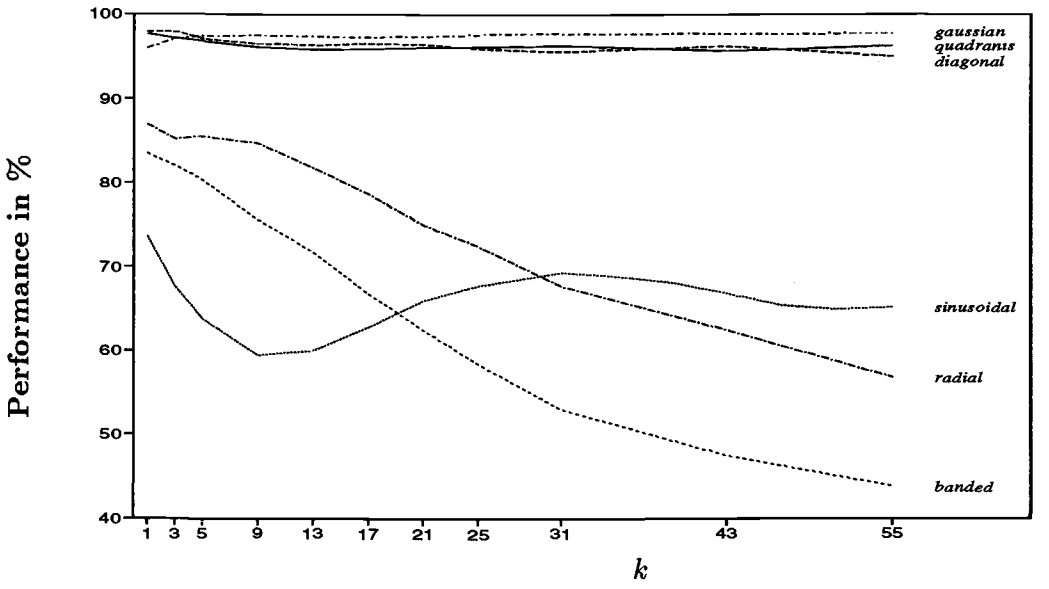


Figure 3.3. The effect of different values of k on the performance of the nearest neighbor algorithm in synthetic data sets (350 training examples, 150 test examples, noise-free).

to the number of training examples of that class with the result that any amount of class noise of less than 50% could be filtered out. However, if (noise-free) examples from different classes are very close together, then k must be kept small to avoid random guessing near the decision boundaries. Hence, a necessary condition for kNN's ability to filter out class noise is that an increase in the value of k leads to a degradation in performance in the noise-free case that is less than the amount of class noise present. In the *quadrants*, *diagonal*, and *gaussian* tasks, any value of k between 1 (3 for the *gaussian* task) and 55 (largest value tested) would lead to a performance indistinguishable from the best observed performance (Figure 3.3, top). In these tasks, class noise can therefore be compensated for to the extent that all non-perturbed test cases that would be classified correctly in the noise-free case would still be classified correctly. In the *banded* and *radial* tasks, k could not be increased beyond 5 without a significant loss in predictive accuracy. With such a small value, only a few noisy training exemplars can be filtered out, which explains why kNN was not able to achieve as significant a performance improvement in these tasks as it achieved in the *quadrants*, *diagonal*, and *gaussian* tasks. The decrease

in performance for k-values between 2 and 13 in the *sinusoidal* task is even more significant than in the *banded* and *radial* tasks. In this task, performance improves again when k is larger than 13. This behavior is due to the distribution of examples from the two classes in the *sinusoidal* task (see also Figure 2.1 and below).

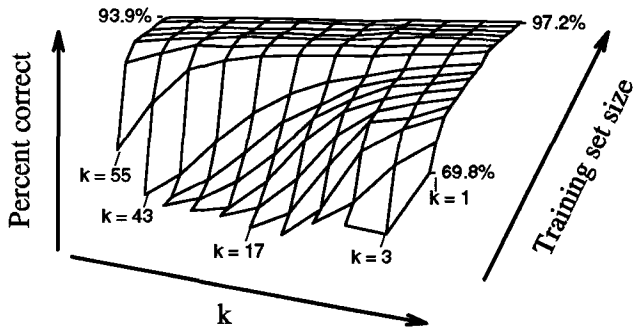


Figure 3.4. The performance of kNN (vertical axis) for different values of k (horizontal axis, k = 1, 3, 5, 9, 13, 17, 21, 25, 31, 43, 55) in the *quadrants* task (noise-free). Sizes of training sets were as follows: 5, 10, 15, 20, 25, 30, 35, 40, 50, 100, 200, 250, 300, 400, 500. Test set size was kept fixed at 150 examples.

The results displayed in Figures 3.2 and 3.3 were obtained from experiments with the standard size data sets as reported in Table 2.1. Experiments with various sizes of training sets yielded the same qualitative results with one exception. For very small training sets (< 100 examples), $k = 1$ always leads to the best performance, independent of the task or level of class noise (Figures 3.4 through 3.7).

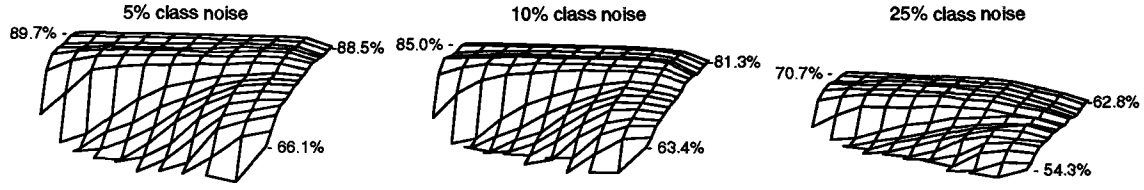


Figure 3.5. The performance of kNN for different values of k in the *quadrants* task in the presence of different levels of class noise. Values on axes are as indicated in Figure 3.4.

Figure 3.6 indicates that the "valley" shown in Figure 3.3 for the *sinusoidal* task can be observed for all training sets with more than 100 examples. This "valley"

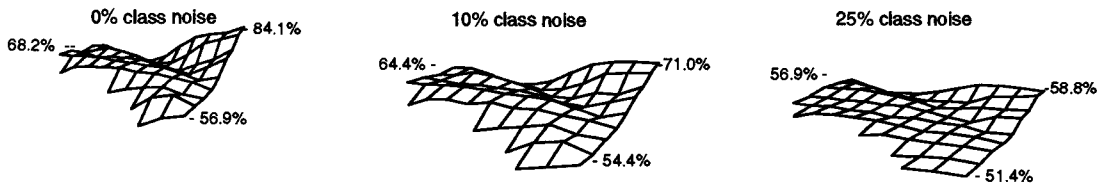


Figure 3.6. The performance of kNN for different values of k in the *sinusoidal* task in the presence of different levels of class noise. Sizes of training sets were 5, 10, 20, 50, 100, 200, 350, 500, 850. All other values are as indicated in Figure 3.4.

is caused by the shape of the decision boundary in the *sinusoidal* task (see also Figure 2.1). The performance in this task initially degrades as k is increased, since an increasing number of examples from the neighboring two "peaks" are used to classify the query. It improves again when k becomes large enough that examples from the four neighboring "peaks" are used. The valley moves toward larger values of k as the training sets become larger (Figure 3.6). This indicates that for large training sets ($>$ 500 examples), the *sinusoidal* task should behave as the *banded* and *radial* tasks, which it does.

Similar results were also obtained when kNN and NN were trained and tested on the letters B and R of the Letter Recognition domain. In this task, kNN outperforms NN only if the training set is relatively large and moderate amounts of class noise are present in the data (Figure 3.7). For very large amounts of noise, performance of both kNN and NN deteriorates rapidly in this domain as well as in all of the constructed tasks. The result is random guessing at a class noise level of $100/n\%$ where n is the number of classes.

To summarize, kNN can overcome moderate amounts of class noise if the centers of different clusters of the noise-free data for the different classes are relatively well separated when compared to their intra-cluster variances, and as long as the sample size is moderate-to-large.

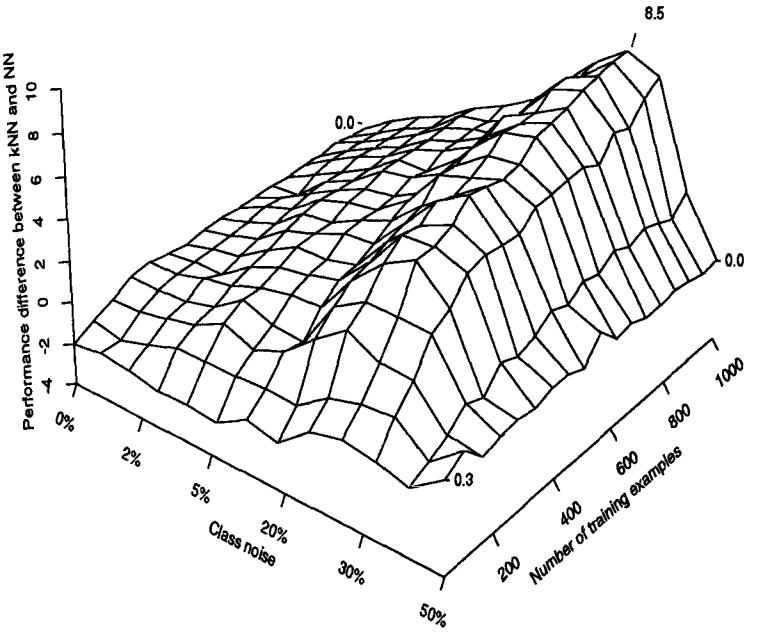


Figure 3.7. Percentage point performance differences between kNN and NN when trained on various combinations of training sets sizes and class noise levels. The letters **B** and **R** of the Letter Recognition domain were used as the evaluation domain. Test set size was kept fixed at 500 examples.

3.1.2 Feature Noise

In the second set of experiments, input feature values were randomly perturbed to simulate noisy feature measurement. A random number within the range indicated in the legend of Figure 3.8 was added to each of the initial input features.[8] Addition of feature noise has an effect similar to class noise on the performance of kNN and NN: performance drops with increasing amounts of noise (Figure 3.8). In experiments with moderate amounts of feature noise, kNN also significantly outperformed NN in most tasks. Two results, however, stand in contrast to what we observed in the class noise experiments: a) There was no level of feature noise where kNN significantly outperformed NN in the *banded* task. b) In the *sinusoidal* task, kNN outperformed NN for nearly any amount of feature noise and all training sets with more than 50 training examples. This varying behavior can be explained by the different effects

[8]However, feature values were not permitted to fall outside the range [0...1].

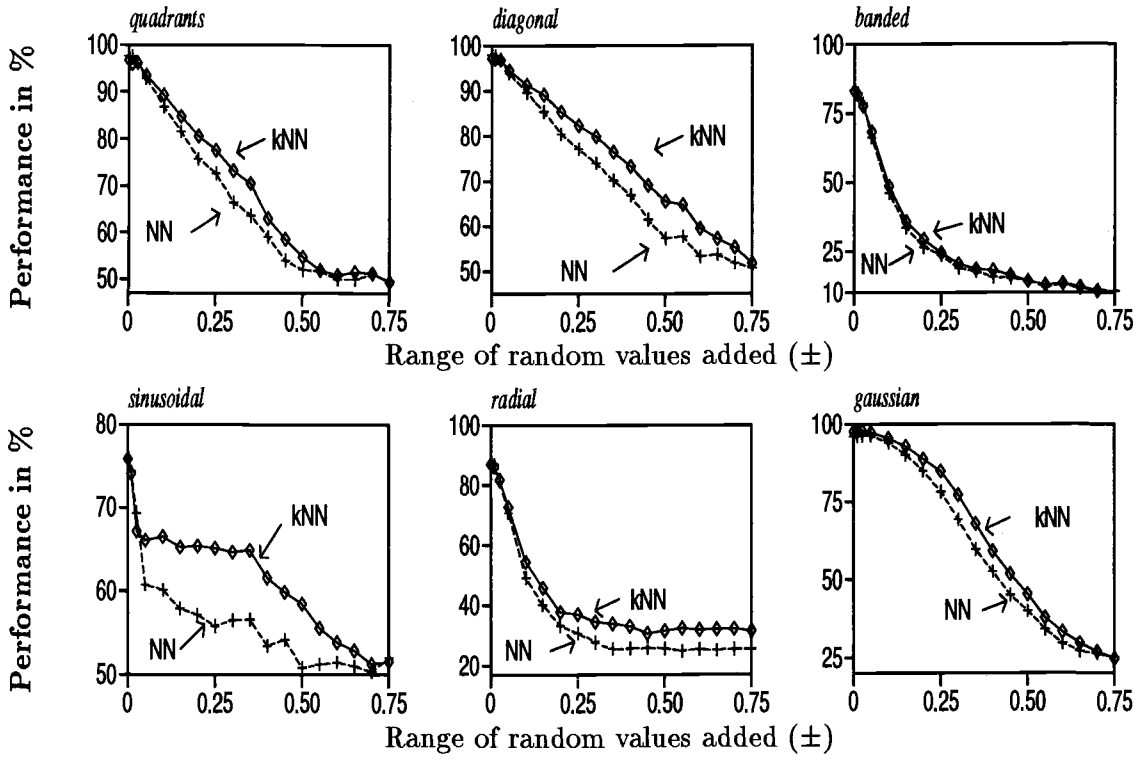


Figure 3.8. The performance of NN and kNN in response to increasing levels of feature noise. The value of x on the horizontal axis indicates that random values in the range $[-x \ldots + x]$ were added to the original features.

class noise and feature noise have on the data. Feature noise can be interpreted as class noise that is restricted to the vicinity of decision boundaries, since feature noise applied to examples distant from the decision boundaries cannot move the examples across a decision boundary. Feature noise applied to examples close to the decision boundary, on the other hand, can move the example to the other side of the boundary, which produces the same effect as flipping the class label of an example on that side of the decision boundary.

If the relative distance from the decision boundary of a significant number of examples from one class is large, then large values of k can be used to eliminate the detrimental effects of some of the "invading" noisy exemplars from other classes. If, on the other hand, there are no regions that remain noise-free when feature noise is added, then larger values of k will quickly reach into areas with examples from other classes. This effect is most apparent in the *banded* task with its narrow bands

that contain examples from the different classes. The kNN algorithm is not able to outperform NN in this task, since addition of feature noise causes the bands containing clean data to become even narrower. A large value of k would then very likely include a large portion of noisy exemplars.

Additional support for this is provided by the following experiment: When the *banded* task was changed to a two class problem (where the two classes occupied alternating bands of data (Figure 2.1)), then kNN also never outperformed NN in the presence of *class* noise. This shows clearly that feature and class noise will have similar effects in tasks where data are distributed in a highly non-Gaussian manner.

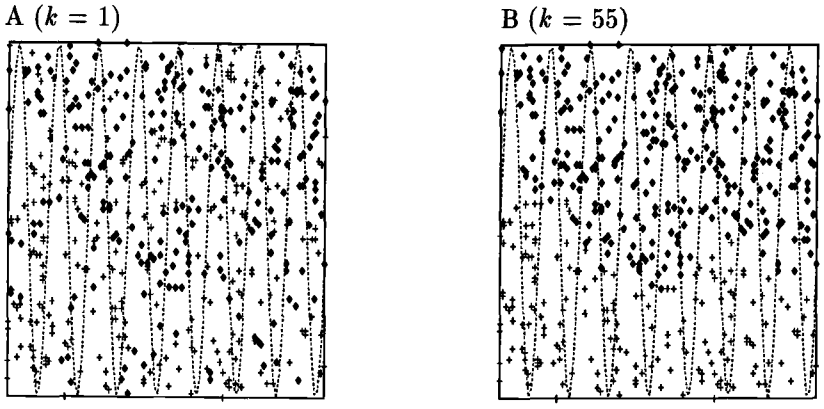


Figure 3.9. The classification labels assigned to 500 test cases in the *sinusoidal* task by the nearest neighbor algorithm. Classifications were computed with $k = 1$ (58.0% correct) in A and with $k = 55$ (66.6% correct) in B. Random values in the range $[-0.1 \ldots + 0.1]$ were added to input features of training and testing cases (350 training examples). The original decision boundary is also plotted for comparison (dotted line).

The different behavior of kNN for class and feature noise in the *sinusoidal* task is due to the fact that significantly more examples belong to class 1 than to class 2 in the upper portion of the input space due to the shape of the sine curve (and vice versa for the lower portion of the input space, Figure 2.1). Cross-validation on k selected values of k between 13 and 67 (median 50) for feature noise in the range -0.1 to $+0.1$. With such large values of k, the decision boundary is a jagged line through the middle of the input space (Figure 3.9). The narrow ends of the sine curve are

therefore smoothed out. This smoothing has a beneficial effect, since many of the examples in these narrow tips are noisy. Such smoothing did not lead to improved performance in the presence of class noise, because, in addition to the few examples within the tips of the sine curve, all data points that were mislabeled due to class noise were also misclassified.

Only small differences in performance between NN and kNN are observed in the *gaussian* task, since in this task feature noise has the effect of generating regions containing examples with random class labels in between the four cluster centers. These regions grow as the level of feature noise is increased. Hence there are only regions that are either relatively noise-free or nearly entirely random. No algorithm can make a good prediction in regions with completely random examples, and previous experiments have shown that NN and kNN will give nearly identical predictions in noise-free regions. A similar argument holds also in the *quadrants* and *diagonal* tasks. This argument qualifies the statement made previously: Although it would be possible to increase the value of k in the *quadrants*, *diagonal*, and *gaussian* tasks, there is little to gain from doing so, since, in the presence of feature noise, regions either contain nearly noise-free examples or nearly entirely noisy examples. In other words, an increase in k has the effect of "sacrificing" small pockets of data belonging to different classes so that the larger areas can be assigned to the proper class in the presence of noise. Naturally, this process only succeeds if significantly more examples can be classified correctly within the larger, smoothed region than must be sacrificed to obtain that larger region.

3.1.3 Irrelevant Features

The third set of experiments involved addition of irrelevant features to the original features of the data sets (Figure 3.10). Irrelevant features were given uniformly distributed random values. Results are very similar to those obtained in the feature noise experiments with one deviation: Large differences in performance are observed in tasks with relatively large regions that are dominated by examples from a single

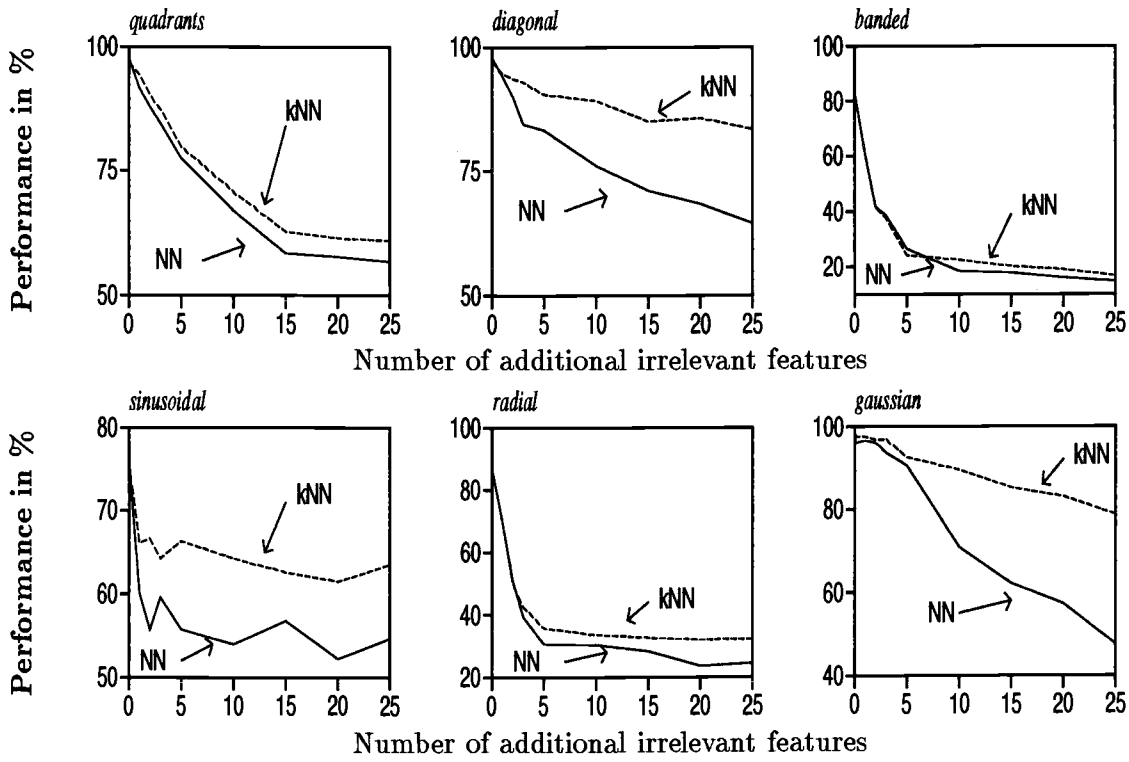


Figure 3.10. The performance of kNN and NN in response to increasing numbers of irrelevant features added to original data points (350 training examples, 150 test examples).

class (the *diagonal* and *gaussian* tasks). The effect is less pronounced in the *quadrants* task than in the *gaussian* task, since in the *quadrants* task more examples are located close to the decision boundary than in the *gaussian* task. Consider, for example, the two pictures in Figure 3.11. The picture on the left side exemplifies the *quadrants* task. Example c is misclassified by any value of k when the irrelevant feature is added. However, in the picture on the right hand side, which exemplifies the *gaussian* task, example c can be classified correctly if $k = 3$. This indicates that larger values of k can be used to compensate for the negative effects of irrelevant features in domains where most examples are well separated from the decision boundary.

To summarize, there is considerable evidence that the often observed superior performance of kNN over NN is due to the presence of moderate amounts of noise in the data. However, larger values of k only improve the classification accuracy of

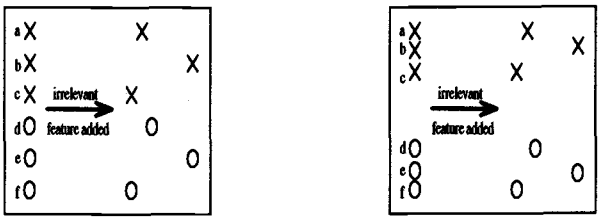


Figure 3.11. An example of the different effects resulting from addition of a single irrelevant feature. The picture on the left exemplifies tasks where examples are close to the decision boundary. Example c would be classified correctly by $k = 1$ when only the relevant feature is considered, but would be misclassified after the irrelevant feature is added. The picture on the right shows a task where examples are more distant from the decision boundary. In that case, example c would still be classified correctly after the irrelevant feature is added.

nearest neighbor algorithms if the training set is relatively large and the noise level is moderate. For very small samples, it is generally best to use $k = 1$, regardless of the level of noise. Also, the effects of class noise and feature noise can be different in certain applications. In particular, class noise cannot be compensated for by larger values of k in tasks with highly curved decision boundaries. While feature noise can only be compensated for in tasks where small pockets with data from one class reach into large regions with data from another class. Irrelevant features have effects similar to feature noise on the relative performance of NN and kNN with the exception that cross-validation on k can lead to an improved performance if there are relatively large (nearly Gaussian) regions containing examples from a single class.

3.2 H2: Choice of Distance Function.

We have already concluded from the experiments with irrelevant features that larger k-values may be used to diminish the negative effects of irrelevant features. A perfect distance function would ignore these features. One could therefore interpret these results as evidence for hypothesis H2, which states that larger k values may be necessary for best performance if the distance function is not well chosen.

Through inspection of the graphs in Figure 2.1, one can easily see that a Euclidean distance metric is not the optimal distance metric in some of the tasks. The

horizontal dimension in the *banded* task, for example, is irrelevant, while the vertical dimension in the *sinusoidal* task is significantly less important than the horizontal dimension. Experiments with a weighted Euclidean distance metric were conducted. Weights were varied from uniform weights to weights that were nearly optimal for each task. As the weighted metric improved, performance of kNN and NN improved (Figure 3.12). However, H2 predicts that the relative improvement in performance of NN should be larger than the relative improvement for kNN. That effect was not observed in the experiments conducted.

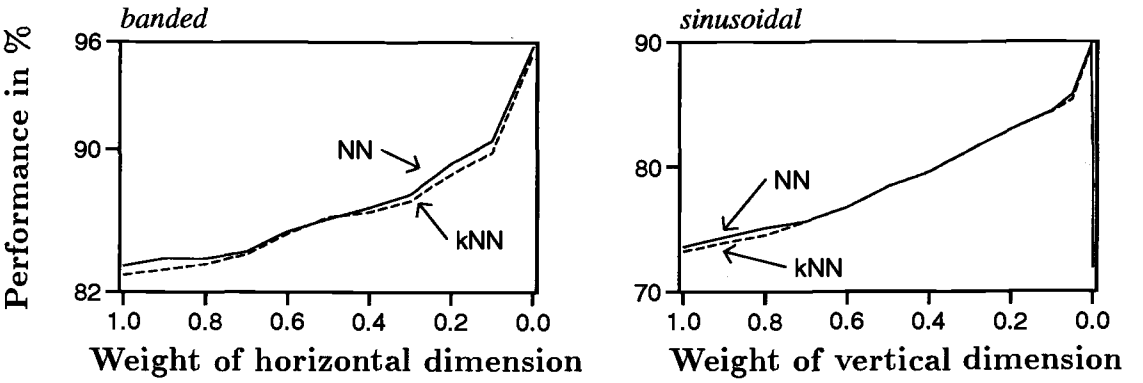


Figure 3.12. The performance of NN and kNN in the *banded* and *sinusoidal* tasks. The change in performance is shown as the weights of the horizontal dimension in the *banded* task and the vertical dimension in the *sinusoidal* task are changed from 1.0 (standard Euclidean distance) to 0.0 (feature ignored). The drop in performance in the *sinusoidal* task for weights of the vertical dimension that are smaller than 0.001 is caused by the fact that then many examples from the two classes have identical input vectors.

Similar results were obtained when the opposite experiment was conducted, that is, when a good metric was replaced by an inferior metric. In the *quadrants* task, both dimensions are equally important. When the weight of the horizontal dimension in the *quadrants* task was slowly decreased to 0, the absolute performance of NN and kNN dropped while the relative performance difference between NN and kNN increased (with NN always being better). Furthermore, no assignment of random weights to the input features could be found in fifteen trials in the Letter Recognition

domain (letters B and R only) such that kNN had a higher classification accuracy than NN.

The above experiments strongly support the conclusion that, contrary to H2, kNN is more susceptible to an improperly weighted Euclidean distance metric than NN (in the noise-free case). However, H2 may still hold if the proper distance metric is not Euclidean. The *radial* task can be used to test H2 in that case. A better distance function for the *radial* task is the "Radial" distance function: $d = |d_1 - d_2|$ where $d_i = \sqrt{(x_i - 0.5)^2 + (y_i - 0.5)^2}$. Figure 3.13 shows for the *radial* task what has been observed in all experiments: as the metric is slowly improved (from Euclidean to the (nearly) optimal "Radial" distance metric, in this case) performance of both algorithms also improves steadily. Furthermore, the relative performance differences between NN and kNN decrease in all experiments as the metric is improved.

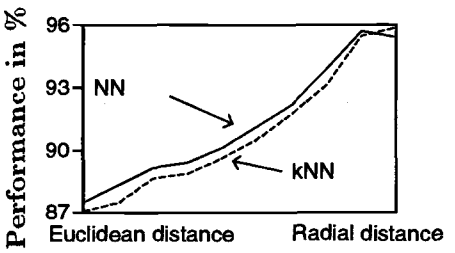


Figure 3.13. The performance of NN and kNN in the *radial* task as the distance metric is changed from Euclidean to Radial.

We conclude that larger values of k cannot be used to compensate for the effects of a non-optimal distance metric. Section 4.3 contains a further discussion of the various distance metrics that can be used for nearest neighbor type algorithms.

3.3 Effect of the Training Set Size

Nearly all experiments described in this chapter have been conducted on a variety of training set sizes. Not surprisingly, the performance of NN and kNN improves as the size of the training sets is increased (see, for example, Figure 3.5). However, it is important to note that in those experiments where kNN outperforms NN, this was

only observed for sufficiently large training sets. The necessary size of the training set varies with the complexity of the given task. One factor that influences the complexity of a task is the relative length of the boundaries between classes. The longer these boundaries are, the more training examples will be necessary to achieve a desired predictive accuracy. This increase in the training set size can be compared to an increase in the number of interpolation points and degree of the spline function used during spline interpolation. The higher the degree of the spline function, the closer the resulting spline will approximate the curve built by the original data. However, in contrast to spline functions, nearest neighbor algorithms can avoid overtraining by increasing the value of k.

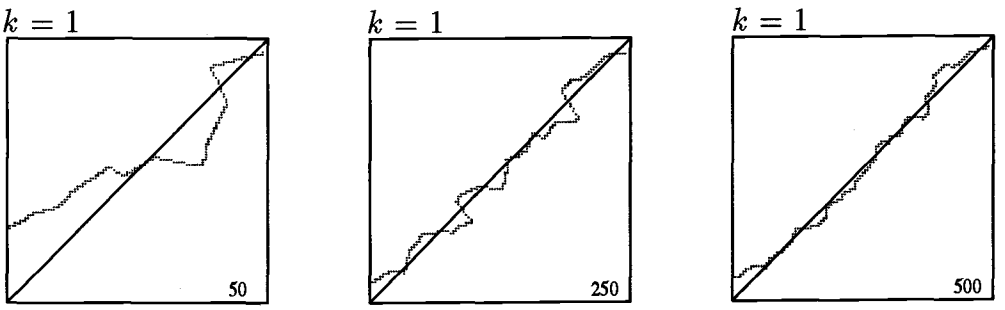


Figure 3.14. The decision boundary produced by NN. The plots show decision boundaries for different sizes of training sets in the *diagonal* task. Numbers in the lower right corner of each plot indicate the size of the training set used to produce each plot.

This point is illustrated in Figures 3.14 through 3.17. In the noise-free case, NN and kNN ($k = 15$) produce similar decision boundaries for large training sets (Figures 3.14 and 3.15). For small data sets, however, the larger divergence of kNN from the correct decision boundary at the extremes of the input space is noticeable. This larger divergence illustrates why kNN's performance is often inferior to NN's when training sets are small: As data points become increasingly more sparse, it is less likely that kNN will find the correct decision boundary, since points further away from the decision boundary influence the decision made. When class noise is present in the data, the advantage of using larger values of k becomes apparent: while first

nearest neighbor creates many pockets with incorrect classifications (Figure 3.16), kNN produces decision boundaries that are very similar to those generated in the noise free case (Figures 3.15 and 3.17).

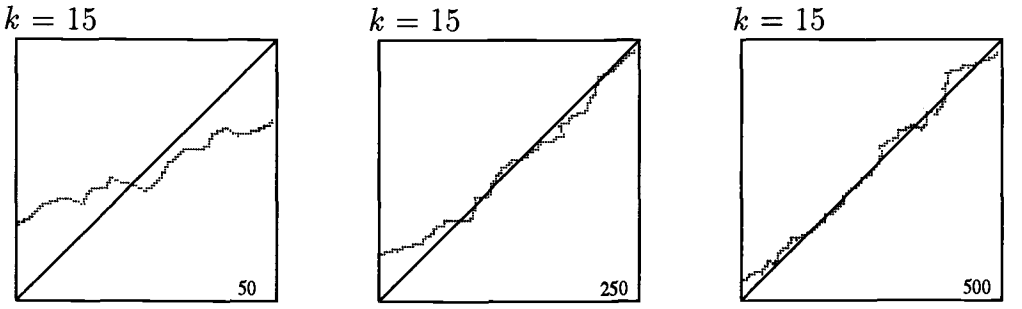


Figure 3.15. The decision boundary produced by kNN ($k = 15$). The plots show decision boundaries for different sizes of training sets in the *diagonal* task. Numbers in the lower right corner of each plot indicate the size of the training set used to produce each plot.

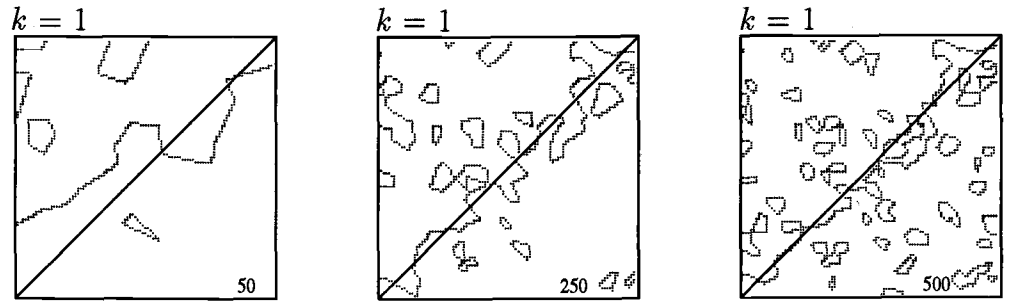


Figure 3.16. The decision boundary produced by NN in the presence of 15% class noise. The plots show decision boundaries for different sizes of training sets in the *diagonal* task. Numbers in the lower right corner of each plot indicate the size of the training set used to produce each plot.

Note that due to the locality of the nearest neighbor algorithm, only the local shape of the decision boundary influences the behavior of the algorithm in that area. It is therefore reasonable to believe that the synthetic data sets used in this section are representative of "real" data sets, since these "real" data sets could be partitioned and re-scaled until they resemble one of the synthetic data sets.

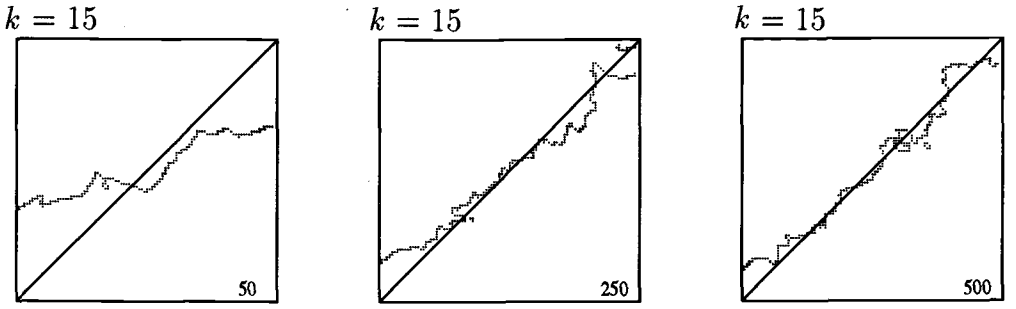


Figure 3.17. The decision boundary produced by kNN ($k = 15$) in the presence of 15% class noise. The plots show decision boundaries for different sizes of training sets in the *diagonal* task. Numbers in the lower right corner of each plot indicate the size of the training set used to produce each plot.

3.4 Summary

The results obtained from the experiments described in this chapter support the following conclusions:

- For very small training sets (< 200 examples) one should always use $k = 1$.
- If the task is known to be noise free and decision boundaries do not overlap, $k = 1$ should be used.
- Irrelevant features have effects similar to feature noise on the performance of the first nearest neighbor and the k-nearest neighbor algorithms.
- Data that are distributed in a highly non-Gaussian manner also necessitate $k = 1$.
- The choice of distance function heavily influences the behavior of NN and kNN. First nearest neighbor is more likely to outperform kNN as the distance metric degrades.
- Larger values of k can lead to significantly improved performance in the presence of moderate amounts of noise for specific data sets. The examples of the different classes of these data sets must belong to clusters where the variances among the different dimensions within each cluster are similar.

Chapter 4
A Study of k-Nearest Neighbor Algorithms

We have seen in the previous chapter that the k-nearest neighbor algorithm can outperform the first-nearest neighbor algorithm on specific data sets. We believe that the characteristics of these data sets (presence of moderate amounts of noise, data from different classes belonging to relatively large Gaussian clusters) may be similar to the characteristics of many data sets encountered in real applications. This chapter is, therefore, devoted to studying methods and modifications of kNN that can improve kNN's performance.

First, we will investigate the issue of how to estimate the value of k that will maximize the predictive accuracy of kNN on the test set. This is followed by a comparison of kNN with simple majority voting versus kNN with weighted voting where the distances of the neighbors determine their weight during voting. Two well known shortcomings of nearest neighbor algorithms are their dependence on the proper choice of the distance metric and their degradation in performance (in accuracy as well as speed) as the number of input dimensions rises. In the second part of this chapter, principal component analysis is evaluated as a means for reducing the number of input dimension of a given task. Finally, two methods that can be used to determine the weights of input features are introduced and compared to each other, as well as to kNN without feature weights.

4.1 Estimating the Value of k

The previous chapter was concerned with whether any value of $k \neq 1$ would lead to a performance superior to that of the first nearest neighbor algorithm. In this section, we will investigate the issue of how the value of k that would lead to the

best performance can be reliably estimated. One assumption that has been made throughout all research reported in the literature to this date is that a single value of k suffices to classify all queries. We will also make that assumption in Section 4.1.1. In Section 4.1.2, however, we will investigate methods that compute different values of k, depending on the local distributions of the data.

4.1.1 Global Determination of k

The experiments described in Section 3 indicate that, as class noise is added, kNN is more likely to outperform NN. These experiments suggest that an increase in the value of k can often be used to compensate for some of the negative effects of noise in the data. In this section, we will investigate the effect that the addition of class noise has on the optimal value of k, how that value can be chosen, and how sensitive performance is to the proper choice of k. We will assume in this section that training sets are large enough so that small sample effects, such as those observed in Figures 3.5 and 3.6, do not affect the experiments. See Weiss [Wei91] for a comparison of different cross-validation methods for kNN classifiers for very small training sets.

Consider again the synthetic data sets described in Figure 2.1. We can determine how the optimal value of k changes as the noise level of data sets increases through the following experiment. Randomly permute the class labels of an increasing number of examples of these tasks. For each level of class noise and for each task, measure the value of k or the range of values of k that would give the best performance. The optimal value of k follows two distinct patterns for the six synthetic data sets (Figure 4.1). In tasks that contain relatively large, near-Gaussian clusters (the *quadrants*, *diagonal*, and *gaussian* tasks), the minimum value of k that gives the best performance rises slowly as the level of class noise increases. The largest value of k that would give the best performance, however, rises very quickly. Hence for moderate amounts of class noise, there is a large range of k values that we can choose from to obtain the best performance. This and the fact that these ranges

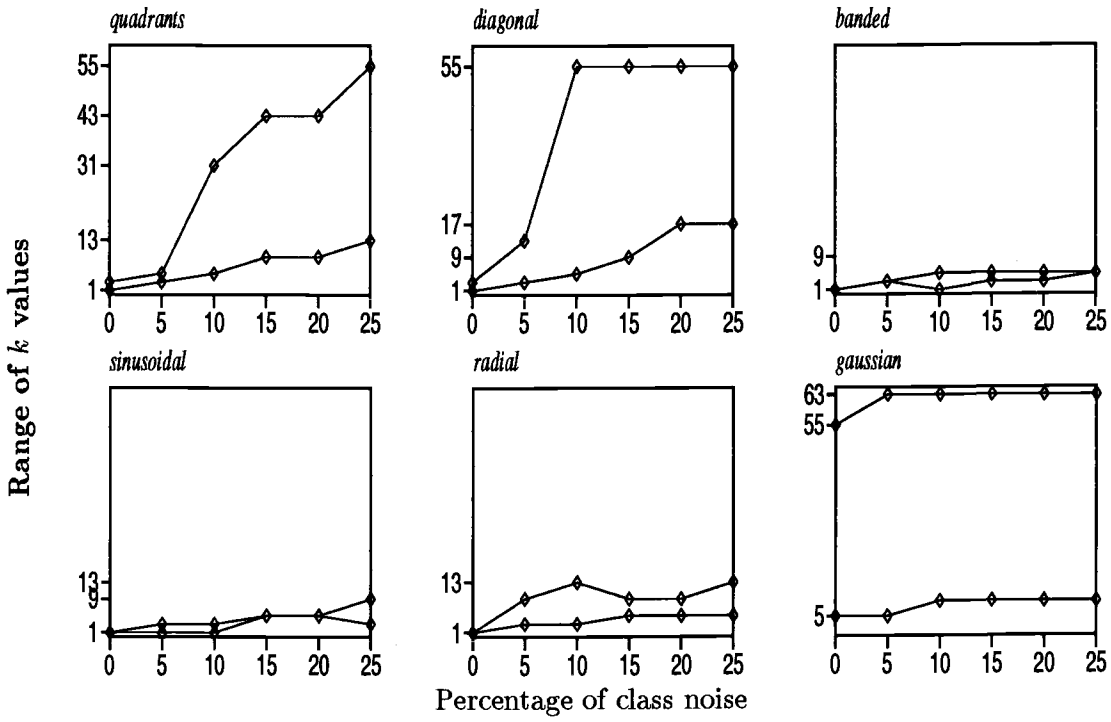


Figure 4.1. Ranges of values of k that resulted in best performance in the synthetic data sets. Any values of k between the upper and lower curves in each graph resulted in a performance within one standard error of the best observed performance (sizes of training sets: 500 (850 in the *sinusoidal* task), test sets: 150).

do not include $k = 1$, indicates once again why kNN can significantly outperform NN in tasks with these characteristics. In the *banded*, *sinusoidal*, and *radial* tasks, on the other hand, there is a very small range of relatively small values of k that would result in the best performance. Such small values of k cannot be used to compensate for any significant amount of noise, which is why in these tasks kNN never outperformed NN.

The large range of k values that give the best performance in tasks where kNN is able to outperform NN indicates that the performance of kNN is not sensitive to the exact choice of k if k is chosen to be large. This suggests that it may not be necessary to consider all possible values of k during cross-validation to obtain the best performance. A restriction in the number of values considered for k during cross-validation has only minor impact on the training time of kNN, since we must still compute all distances which is by far the most expensive operation during cross-

validation. Nonetheless, this restriction may result in a more robust behavior of kNN and it may help characterizing data sets by the specific value of k that was chosen by kNN.

Choosing a Cross-validation Method Even when we restrict the number of values k can assume, the issue of how to choose that value still remains. The optimal method would give the most reliable estimate with the smallest computational effort. We will use the synthetic data sets and some other data sets described in Section 2.2 to evaluate several methods for estimating the value of k. Three methods for choosing the value of k are compared below. (See Table 4.1 for an explanation of terms.)

1. The value of k could be set to some constant value before experimentation (denoted by $\text{kNN}_{k=x}$).

2. Leave-one-out cross-validation (Section 2.4.8) can be used to estimate the optimal value of k (denoted by $\text{kNN}_{cv^{all}_{leave-1-out}}$). This procedure is the computationally most expensive of the methods discussed here.

3. The value of k could also be determined via one-fold cross-validation (denoted by $\text{kNN}_{cv^{odd}_{1-fold}}$). When compared to $\text{kNN}_{cv^{all}_{leave-1-out}}$, this method requires very little training time.

The experiments indicate that setting k to some fixed value without cross-validation would only lead to good results if the experimenter had some prior knowledge as to what the value of k should be (Tables 4.1 and 4.2). Prior knowledge is necessary since no single value of k gave the best performance in all possible tasks. One choice of k that is often used in the literature is $k = 1$. A choice of $k = 1$ may often lead to inferior classification accuracies, since NN performed significantly worse than kNN_{cv} in 7 of the 9 non-synthetic domains (Table 4.2). The performance of $\text{kNN}_{cv^{odd}_{1-fold}}$ is superior to that of the first nearest neighbor in nearly all (8 out of 10, Tables 4.1 and 4.2) tasks that are not entirely noise-free. Furthermore, in five of the nine experiments described in Table 4.2, this method matched the best average

performance that can be achieved with any fixed value of k (this method is denoted by $\text{kNN}_{k=best}$ in Tables 4.1 and 4.2).

When 1-fold cross-validation is conducted on odd values of k only, an accuracy on the separate test sets that was statistically indistinguishable from that of $\text{kNN}_{cv^{all}_{leave-1-out}}$ in 12 of the 15 experiments (Tables 4.1 and 4.2) was achieved. This method was inferior to $\text{kNN}_{cv^{all}_{leave-1-out}}$ in two tasks and superior in the *radial* task.

The results displayed in Tables 4.1 and 4.2 indicate that the sensitivity of kNN to the exact choice of k decreases as k becomes larger.[9] Furthermore, in domains with irrelevant features (Hungarian, Led-24 Display, and Waveform-40 domains) relatively large (> 20) values for k are required for best performance. Leave-one-out cross-validation on all possible values of k generally leads to the best performance across all domains that are not entirely noise-free. However, cross-validation on odd values only ($\text{kNN}_{cv^{odd}_{leave-1-out}}$) leads to accuracies that are statistically indistinguishable from that of $\text{kNN}_{cv^{all}_{leave-1-out}}$. The set of potential candidates for k can even be further reduced without any significant loss in predictive accuracy. When cross-validation was conducted on the arbitrarily chosen values $k = 1$, 3, 17, 31, and 47 alone ($\text{kNN}_{cv^{1,3,17,31,47}_{leave-1-out}}$), performance was never inferior to that of $\text{kNN}_{cv^{all}_{leave-1-out}}$ and superior in the Voting domain.

In summary, to obtain the best performance it is necessary to perform cross-validation on no more than approximately 10 values for k. The consistently good performance of $\text{kNN}_{cv^{1,3,17,31,47}_{leave-1-out}}$ gives strong support for the hypothesis that this will be true in the majority of the applications that are encountered in the real world. In most applications, one-fold cross-validation can be used instead of leave-one-out cross-validation, if the time required to train the classifier is of concern.

Choosing Among Different Candidates for k The determination of the value of k as described above was exclusively governed by the performance of the different

[9]This is indicated by the fact that the range of k values that gave the best performance increases as the smallest k that gave the best performance increases.

Table 4.1. The performance of the k-nearest neighbor algorithm in synthetic tasks (noise-free) for different methods of determining the value of k. Shown is the absolute value ($\pm$ standard error) of the best average performance that can be achieved by any single fixed value of k during 25 repetitions ($kNN_{k=best}$) and the relative performance differences between all other methods and $kNN_{k=best}$. The subscript $k = i$ indicates that k was fixed at i. Numbers in parentheses in the row denoted by $kNN_{k=best}$ indicate the range of k values that resulted in a performance within one standard error of the best performance. The subscript cv indicates cross-validation. The superscript to cv indicates the kind of potential candidates for k, and the subscript indicates the type of cross-validation. Significance of difference to $kNN_{k=1}$ ($kNN_{cv^{all}_{leave-1-out}}$) is indicated by * ($_*$).

Method	*Quadrants* ‡	*Diagonal* ‡	*Banded* ‡
$kNN_{k=1}$	+0.0	+0.0	+0.0
$kNN_{k=3}$	−0.2	−0.1	-1.3^{***}
$kNN_{k=17}$	-1.6^{*****}_{****}	-0.8^{****}_{***}	-9.5^{*****}_{*****}
$kNN_{k=31}$	-2.0^{*****}_{*****}	-0.9^{****}_{**}	-21.6^{*****}_{*****}
$kNN_{k=best}$	97.2±0.3 (1-1)	98.2±0.2 (1-3)	86.9±0.6 (1-1)
$kNN_{cv^{odd}_{1-fold}}$	−0.3	-0.4^{**}	−0.8
$kNN_{cv^{all}_{leave-1-out}}$	−0.2	−0.1	−0.7
$kNN_{cv^{odd}_{leave-1-out}}$	-0.4^{*}	+0.0	−0.7
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	-0.4^{**}	+0.0	−0.5
Method	*Sinusoidal* †	*Radial* ‡	*Gaussian* ‡
$kNN_{k=1}$	+0.0	$+0.0_{**}$	-0.6_{**}
$kNN_{k=3}$	-3.8^{*****}_{*****}	-1.0^{**}	−0.4
$kNN_{k=17}$	-21.3^{*****}_{*****}	-5.4^{*****}_{*****}	$+0.0^{**}$
$kNN_{k=31}$	-26.8^{*****}_{*****}	-25.7^{*****}_{*****}	+0.0
$kNN_{k=best}$	84.1±0.4 (1-1)	$90.3\pm0.6\ (1\text{-}1)_{**}$	$97.9\pm0.2\ (5\text{-}43)^{**}$
$kNN_{cv^{odd}_{1-fold}}$	-1.2^{**}_{**}	$+0.1_{**}$	-0.3^{**}
$kNN_{cv^{all}_{leave-1-out}}$	+0.0	-1.0^{**}	-0.1^{**}
$kNN_{cv^{odd}_{leave-1-out}}$	+0.0	-0.9^{*}	$-0.1\ ^{**}$
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	+0.0	-0.6^{*}	-0.2^{**}

† 850 training examples, 150 test examples

‡ 500 training examples, 150 test examples

$\star\star\star\star\star$ $p < 0.001$, $\star\star\star\star$ $p < 0.005$, $\star\star\star$ $p < 0.01$, $\star\star$ $p < 0.05$, $\star$ $p < 0.1$

Table 4.2. The performance of the k-nearest neighbor algorithm in non-synthetic tasks for different methods of determining the value of k. The meaning of the terms used is the same as in Table 4.1.

Method	Iris	Voting	Cleveland
$kNN_{k=1}$	-1.1	-5.6_{*****}	-6.2_{*****}
$kNN_{k=3}$	-1.0	-0.6^{*****}	-1.8^{*****}_{*}
$kNN_{k=17}$	-0.8	-1.9^{*****}_{*****}	-1.1^{*****}
$kNN_{k=31}$	-3.1^{****}_{****}	-2.7^{*****}_{*****}	-1.1^{*****}
$kNN_{k=best}$	96.3±0.4 (5-13)**	92.5±0.4 (5-9)$^{*****}_{**}$	84.0±0.5 (19-57)*****
$kNN_{cv^{odd}_{1-fold}}$	-1.1	-0.7^{*****}	-1.5^{*****}
$kNN_{cv^{all}_{leave-1-out}}$	-0.7	-0.5^{*****}	-0.6^{*****}
$kNN_{cv^{odd}_{leave-1-out}}$	-0.7	-0.3^{*****}_{*}	-0.9^{*****}
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	-1.3	-0.2^{*****}_{***}	-0.8^{*****}
Method	Hungarian	Isolet	Led-7 Display
$kNN_{k=1}$	-7.9_{*****}	-0.9_{*}	-2.1_{*****}
$kNN_{k=3}$	-4.6^{*****}_{*****}	-0.6	-0.2^{*****}
$kNN_{k=17}$	-2.3^{*****}	-0.4	-5.8^{*****}_{*****}
$kNN_{k=31}$	-1.8^{*****}	-1.7^{**}_{*****}	-13.6^{*}_{*****}
$kNN_{k=best}$	83.8±0.9 (37-71)$^{*****}_{***}$	84.0±0.4 (5-13)$^{***}_{**}$	72.6±0.5 (3-5)*****
$kNN_{cv^{odd}_{1-fold}}$	-2.9^{*****}	-0.6	-0.6^{*}_{*****}
$kNN_{cv^{all}_{leave-1-out}}$	-1.8^{*****}	-0.4^{*}	-0.3^{*****}
$kNN_{cv^{odd}_{leave-1-out}}$	-1.7^{*****}	-0.4^{**}	-0.2^{*****}
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	-1.2^{*****}	-0.2^{**}	-0.7^{*****}_{*}
Method	Led-24 Display	Waveform-21	Waveform-40
$kNN_{k=1}$	-21.4_{*****}	-6.9_{*****}	-10.3_{*****}
$kNN_{k=3}$	-13.7^{*****}_{*****}	-3.9^{****}_{*****}	-6.5^{*****}_{*****}
$kNN_{k=17}$	-2.8^{*****}_{*****}	-0.3^{*****}	-1.1^{*****}
$kNN_{k=31}$	-1.0^{*****}	-0.2^{*****}	$+0.0^{*****}$
$kNN_{k=best}$	69.9±0.6 (43-67)$^{*****}_{**}$	82.1±0.8 (17-55)*****	79.7±0.8 (21-55)*****
$kNN_{cv^{odd}_{1-fold}}$	-1.6^{*****}_{**}	-0.5^{*****}	-0.1^{*****}
$kNN_{cv^{all}_{leave-1-out}}$	-0.5^{*****}	-0.2^{*****}	$+1.0^{*****}$
$kNN_{cv^{odd}_{leave-1-out}}$	-0.4^{*****}	-0.3^{*****}	$+1.1^{*****}$
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	-0.3^{*****}	-0.5^{*****}	$+0.7^{*****}$

candidates for k during cross-validation. The actual value of k only played a role when two or more values of k gave the same performance. The algorithm as used in Section 4.1.1 prefers the smaller k when the performance is equal. It is well known that smaller values of k lead to larger variance (i.e. local idiosyncrasies of the data have a large influence on the output of the classifier) while larger values employ a stronger bias [GBD92]. According to Geman et al [GBD92], a strong bias may be necessary for optimal performance when the available training set is relatively small. However, this stands in contrast to the results that we observed in Chapter 3 that indicate that one should always use $k = 1$ for very small data sets. This apparent contradiction may be caused by the fact that for very small data sets any bias that is stronger than the bias of NN may already be too strong.

The experiments described above have shown that the performance of kNN is not sensitive to the exact choice of k when k is large. We would expect, therefore, that a modification of kNN's preference bias has little effect on its performance. The following modifications were tested to evaluate this hypothesis:

- Choose the smallest k that gave an accuracy within one percentage point of the best observed cross-validation performance (denoted by $\text{kNN}_{min\ k-1}$).
- Choose the smallest k with the best cross-validation performance (this is the original version, denoted by $\text{kNN}_{min\ k}$).
- Choose the largest k with the best cross-validation performance (denoted by $\text{kNN}_{max\ k}$).
- Choose the largest k that gave an accuracy within one percentage point of the best observed cross-validation performance (denoted by $\text{kNN}_{max\ k-1}$).
- Minimize the risk of choosing the wrong k through smoothing (denoted by $\text{kNN}_{min\ risk\ k}$). This can be achieved by choosing $k = j$ to minimize $0.1 \cdot \text{P}(\text{error}_{j-2}) + 0.2 \cdot \text{P}(\text{error}_{j-1}) + 0.4 \cdot \text{P}(\text{error}_j) + 0.2 \cdot \text{P}(\text{error}_{j+1}) + 0.1 \times \text{P}(\text{error}_{j+2})$ (where $\text{P}(\text{error}_j)$ is the error rate of kNN with $k = j$ as estimated from the training data and $\text{P}(\text{error}_{-1}) = \text{P}(\text{error}_0) = \text{P}(\text{error}_1)$). This

method can be used to filter out coincidental peaks in the cross-validation performance that would otherwise be chosen over the more robust value as indicated by the good performance obtained by the k values that are immediately smaller or larger. Consider Figure 4.2 which shows the leave-one-out cross-validation accuracies obtained in an experiment in the Hungarian domain before and after smoothing. Without smoothing, $k = 29$ would have been chosen and resulted in a performance on the test set of 84.1% correct, while smoothing indicated that $k = 45$ should be used, which gave an accuracy on the test set of 88.6% correct.

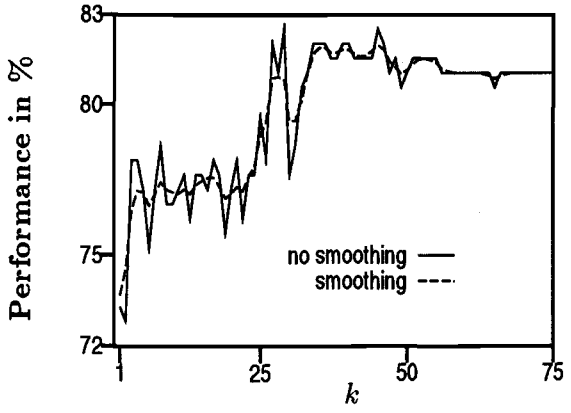


Figure 4.2. Cross-validation accuracy for different values of k in the Hungarian domain. Shown is performance without and with smoothing.

The average classification accuracies are similar for all methods in all domains (as shown in Table 4.3). In the noise-free tasks (the *quadrants*, *diagonal*, *banded*, *sinusoidal*, and *radial* tasks) slightly better results were obtained with smaller ks, while larger values of k give slightly better accuracies in the other domains. The best overall method was $\text{kNN}_{min\ risk\ k}$.

4.1.2 Locally Adaptive Nearest Neighbor Algorithms

The methods discussed previously assumed that a single uniform value for k is used to classify all queries. For certain applications, it might be desirable to vary the

Table 4.3. The performance of the kNN algorithm when different procedures for choosing the value of k are employed. Shown is the absolute percentage correct for $kNN_{min\ k}$ ($\pm$ standard error) and the relative differences in performance of the other methods when compared to $kNN_{min\ k}$.

	kNN				
Domain	*min k−1*	*min k*	*max k*	*max k−1*	*min risk k*
Quadrants	−0.3	96.8±0.4	−0.1	−0.1	+0.2
Diagonal	+0.6	97.2±0.5	+0.1	−0.8	+0.0
Banded	+0.3	83.0±0.7	+0.0	+0.2	+0.5
Sinusoidal	+0.0	73.2±0.7	−0.2	−0.2	−0.5
Radial	+0.1	86.9±0.6	−0.1	−0.7	+0.0
Gaussian	−0.1	97.5±0.2	+0.1	+0.2	−0.1
Iris	−0.2	95.6±0.5	+0.1	−0.4	+0.3
Cleveland	−0.5	83.4±0.5	+0.0	−0.4	−0.1
Hungarian	−0.8	82.1±1.0	+0.3	+1.1	+0.4
Voting	−0.4	92.0±0.4	+0.1	−0.5	+0.3
Waveform-21	−0.6	81.8±0.9	+0.0	+0.1	+0.1
Waveform-40	+0.5	80.7±1.1	+0.2	+0.7	+0.5
Led-7 Display	−0.2	72.3±0.6	+0.5	+0.5	+0.0
Led-24 Display	−0.5	69.4±0.6	+0.1	+0.1	+0.2
Letter Recognition$_{BR}$	+0.1	97.5±0.1	+0.3	−0.5	+0.2

$kNN_{min\ k-1}$: The smallest value of k that gave a cross-validation performance within one percentage point of the best observed cross-validation performance was chosen.
$kNN_{min\ k}$: In case of ties, the smallest value that gave the best cross-validation performance was chosen.
$kNN_{max\ k}$: In case of ties, the largest value that gave the best cross-validation performance was chosen.
$kNN_{max\ k-1}$: The largest value of k that gave a cross-validation performance within one percentage point of the best observed cross-validation performance was chosen.
$kNN_{min\ risk\ k}$: k was chosen to minimize $0.1 \cdot P(error_{k-2}) + 0.2 \cdot P(error_{k-1}) + 0.4 \cdot P(error_k) + 0.2 \cdot P(error_{k+1}) + 0.1 \cdot P(error_{k+2})$ ($P(error_{-1}) = P(error_0) = P(error_1)$).

value of k locally within different parts of the input space to account for varying characteristics of the data such as noise or irrelevant features. However, for lack of an algorithm, researchers have assumed a global value for k in all work concerning nearest neighbor algorithms to date [BV92]. In this section, four new algorithms that determine different values for k in different parts of the input space are proposed and evaluated. The locally varying values of k computed by these algorithms are used to classify novel examples. These four algorithms use different methods to compute the k-values that are used for classification.

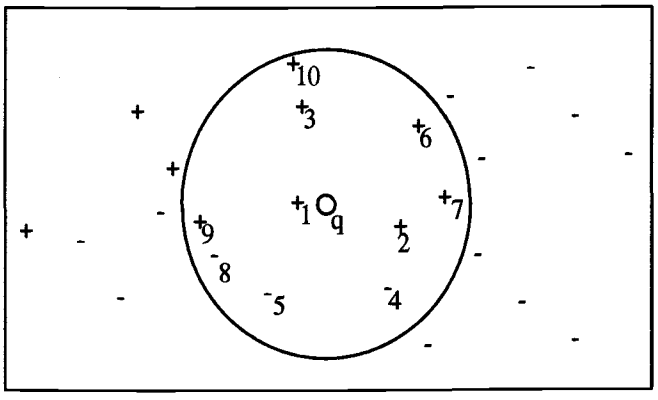
We can split the problem of locally choosing the value of k into two phases. In the first phase, statistics on good values of k in various parts of the input space are gathered at learning time. In the second phase, during classification, a value for k must be chosen based on the location of the query and the previously computed statistics in its immidiate surroundings.

We developed four algorithms based on this idea:

- localKNN$_{ks\ unrestricted}$

  This is the basic local kNN algorithm. The three subsequent algorithms are modifications of this method. This algorithm stores all of the training examples. Along with each training example, it stores a list of those values of k that correctly classify that example under leave-one-out cross-validation. To classify a query q, the M nearest neighbors of the query are computed, and that k which classifies correctly most of these M neighbors is determined. Call this value $k_{M,q}$. The query q is then classified with the class of the majority of its $k_{M,q}$ nearest neighbors. Note that $k_{M,q}$ can be larger or smaller than M. The parameter M is the only parameter of the algorithm, and it can be determined by cross-validation. Consider, for example, Figure 4.3. The location of the query q is indicated by the small cirle in the center of the figure. Assume $M = 10$, the M nearest neighbors of q then lie within the larger circle. The table in Figure 4.3 shows for each of the 10 nearest neighbors of q the set of k values that would result in a correct classification of these neighbors. The value

$k = 5$ would classify correctly 8 of the 10 nearest neighbors of q (as indicated by the last column in the table). We would, therefore, set $k_{10,q}$ to 8. Query q would, thus, be classified as belonging to class +. Note that the table shown in Figure 4.3 can be computed at training time. All we have to do at classification time is to extract the M columns that indicate the good k values of the M nearest neighbors of the query from this global table.



	Pattern number										
k	1	2	3	4	5	6	7	8	9	10	Sum
1		x	x		x					x	4
2			x							x	2
3			x	x				x		x	4
4			x							x	2
5	x	x	x	x	x	x		x		x	8
6			x							x	2
7	x		x	x	x	x		x		x	7
8	x		x	x						x	4
9			x	x	x	x				x	5
10	x		x	x	x					x	5
11	x		x	x	x				x	x	6
12	x		x	x	x					x	5
13	x		x	x	x				x	x	6
14	x		x	x	x				x	x	5
15	x		x	x					x	x	5

Figure 4.3. An example of how localKNN$_{ks\ unrestricted}$ would determine the value of k that should be used to classify query q. The picture shows the 10 nearest neighbors of q and their classification labels. The table contains for each value of k from 1 to 15, and for each of the 10 neighbors of q whether that k would correctly classify the training example (x).

- localKNN$_{ks\ pruned}$

  The list of good k values for each training example generally contains many values. A more robust performance might be achieved if some of the less frequent values in these lists are removed. This can be accomplished as follows. A global histogram of k values is computed, and k values that appear fewer than L times are pruned from all lists (at least one k value must, however, remain in each list). The parameter L can be estimated via cross-validation. Classification of queries is identical to localKNN$_{ks\ unrestricted}$. In the example shown in Figure 4.3, we would compute the sum of each row of the global table of good k values and remove those rows that sum up to less than L.

- localKNN$_{one\ k\ per\ class}$

  For each output class, the value of k that would result in the correct (leave-one-out) classification of the maximum number of training patterns from that class is determined. A query q is classified as follows: Assume there are two output classes, C_1 and C_2. Let k_1 and k_2 be the k value computed for classes C_1 and C_2, respectively. The query is assigned to class C_1 if the percentage of the k_1 nearest neighbors of q that belong to class C_1 is larger than the percentage of the k_2 nearest neighbors of q that belong to class C_2. Otherwise, q is assigned to class C_2. Generalization of that procedure to any number of output classes is straightforward.

- localKNN$_{one\ k\ per\ cluster}$

  An unsupervised clustering algorithm (RPCL [XKO93], see Section 2.4.7), is applied to determine clusters of input data. A single k value is determined for each cluster by leave-one-out cross-validation on all the examples that are part of that cluster. However, the entire training set is used to classify each of these cross-validation examples. Each query is classified according to the k value of the cluster it is assigned to.

Experiments with Constructed Data Sets Three constructed data sets were used to determine the ability of local k-nearest neighbor methods to determine proper values of k. These data sets were constructed such that it was known before experimentation that varying k values should lead to superior performance:

Data set Letter-Led: Previous experiments have shown that in the Letter Recognition domain, very small values of k lead to the best performance. In the Led-24 Display domain, on the other hand, values of k that are typically larger than 50 resulted in the best performance (Table 4.2). The Letter Recognition domain has only 16 input features. We removed 8 of the 17 irrelevant features of the Led-24 Display domain to obtain two data sets with the same number of input features. When trained and tested on these two data sets, local kNN methods performed worse than the global kNN algorithm (Figure 4.4). However, when the Letter Recognition and the Led-16 data sets were combined into a single data set, performance of all local kNN methods was superior to that of kNN (Figure 4.4, "Combined").

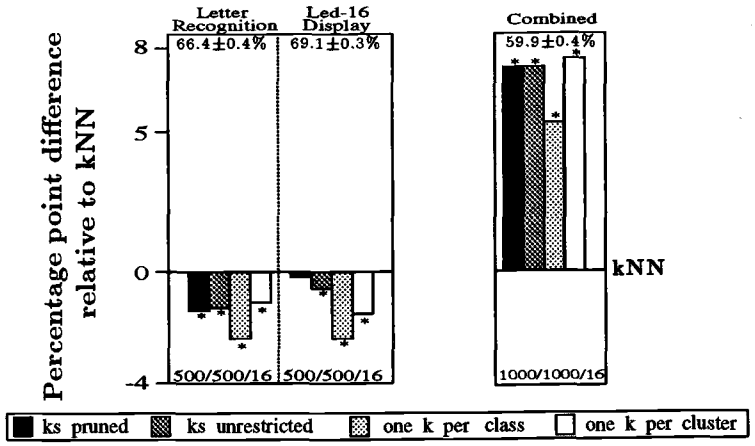


Figure 4.4. Percent accuracy of local kNN methods relative to kNN on the Letter Recognition and the Led-16 Display data sets (left) and on the combination of these data sets (right). These differences (*) were statistically significant ($p < 0.05$). Results are based on 25 repetitions. Shown at the bottom of each graph are sizes of training sets/sizes of test sets/number of input features. The percentage at top of each graph indicates average accuracy of kNN $\pm$ standard error.

Data set Sine-Wave: Data points were drawn from two two-dimensional sine-curves with identical periodicity. The curve representing class 1 was slightly offset in

the vertical direction from the curve representing class 2. This data set requires small values for k ($k \approx 7$) for best performance. Another data set that requires very large values for k is the Waveform-21 data set (Table 4.2). The Waveform-21 data set has 21 input feature. Hence we added 19 features having value 0 to the two-dimensional data set. When trained and tested on these two data sets, all local kNN methods performed slightly better than the global kNN algorithm, the better performance of localKNN$_{one\ k\ per\ cluster}$ was statistically significant (Figure 4.5). When trained and tested on the combination of these data sets, all local kNN methods performed better than kNN on a statistically significant level (Figure 4.5).

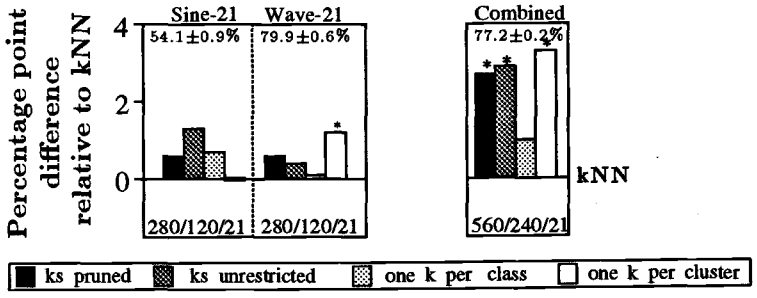


Figure 4.5. Percent accuracy of local kNN methods relative to kNN on the Sine and the Waveform-21 data sets (left) and on the combination of these data sets (right). These differences (*) were statistically significant ($p < 0.05$). Results are based on 25 repetitions. Shown at the bottom of each graph are sizes of training sets/sizes of test sets/number of input features. The percentage at top of each graph indicates average accuracy of kNN $\pm$ standard error.

Data set Two-lines: This data set was constructed to display some characteristics of data sets for which we assumed local kNN methods would work best (Figure 4.6). The data set was constructed such that patterns from two classes were stretched out along two parallel lines in one part of the input space. The parallel lines were spaced such that the nearest neighbor for most patterns belonged to the same class as the pattern itself, while two out of the three nearest neighbors belonged to the other class. In other parts of the input space, classes were well separated, but class labels were flipped such that the nearest neighbor of a query would indicate the wrong pattern, while the majority of the k nearest neighbors ($k > 3$) would

indicate the correct class (see also Figure 4.6). The local kNN methods were able to significantly outperform kNN on this tasks (Figure 4.7).

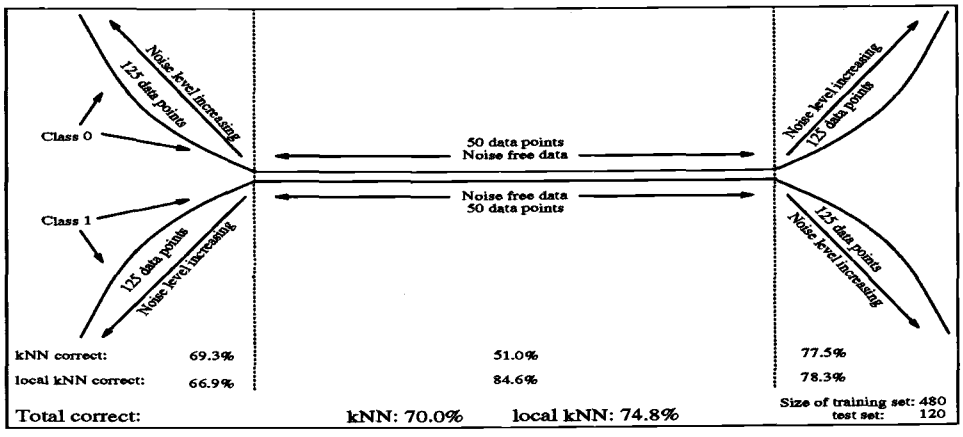


Figure 4.6. Data points for the Two-lines data set were drawn from either of the two displayed curves (i.e. all data points lie on either of the two curves). Class labels were flipped with increasing probabilities to a maximum noise level of approximately 45% at the respective ends of the two lines. Listed at the bottom is performance of kNN and localKNN$_{unrestricted}$ within different regions of the input space and for the entire input space.

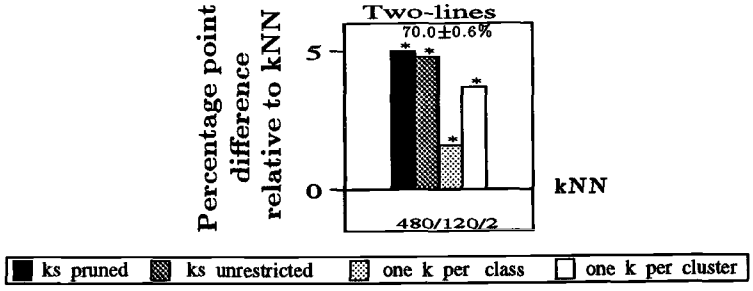


Figure 4.7. Percent accuracy of local kNN methods relative to kNN on the Two-lines data set. These differences (*) were statistically significant ($p < 0.05$). Results are based on 25 repetitions. Shown at the bottom of each graph are sizes of training sets/sizes of test sets/number of input features. The percentage at top of each graph indicates average accuracy of kNN $\pm$ standard error.

These experiments suggest that local kNN methods can lead to significant improvements over kNN in predictive accuracy. The best performing local methods are localKNN$_{ks\ pruned}$, localKNN$_{ks\ unrestricted}$, and localKNN$_{one\ k\ per\ cluster}$. These

methods were outperformed by kNN in two of the original data sets. However, the performance of these methods was clearly superior to kNN in all domains where data were collections of significantly distinct subsets.

Experiments with Commonly Used Data Sets Twelve domains of varying sizes and complexities (c.f. Section 2.2) were employed to compare the performance of the various nearest neighbor algorithms. Results displayed in Figure 4.8 indicate that in most data sets which are commonly used to evaluate machine learning algorithms, local nearest neighbor methods have only minor impact on the performance of kNN. The best local methods are either indistinguishable in performance from kNN (localKNN$_{one\ k\ per\ cluster}$) or inferior in only one domain (localKNN$_{ks\ pruned}$).

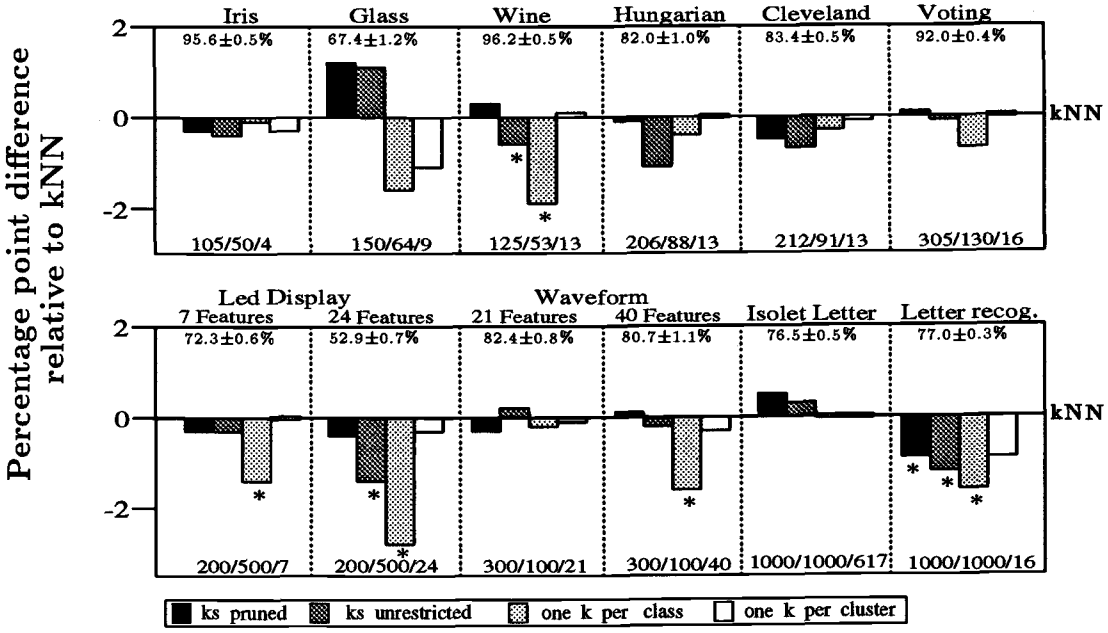


Figure 4.8. Percent accuracy of local kNN methods relative to kNN on separate test sets. These differences (*) were statistically significant ($p < 0.05$). Results are based on 25 repetitions. Shown at the bottom of each graph are sizes of training sets/sizes of test sets/number of input features. The percentage at top of each graph indicates average accuracy of kNN $\pm$ standard error.

The number of actual k values selected varies significantly for the different local methods (Table 4.4). Not surprisingly, localKNN$_{ks\ unrestricted}$ employs the largest

number of distinct k values in all domains. Pruning of ks significantly reduced the number of values used in all domains. However, the method using the fewest distinct k values is localKNN$_{one\ k\ per\ cluster}$, which also explains the similar performance of kNN and localKNN$_{one\ k\ per\ cluster}$ in most domains. Note that several clusters computed by localKNN$_{one\ k\ per\ cluster}$ may use the same k.

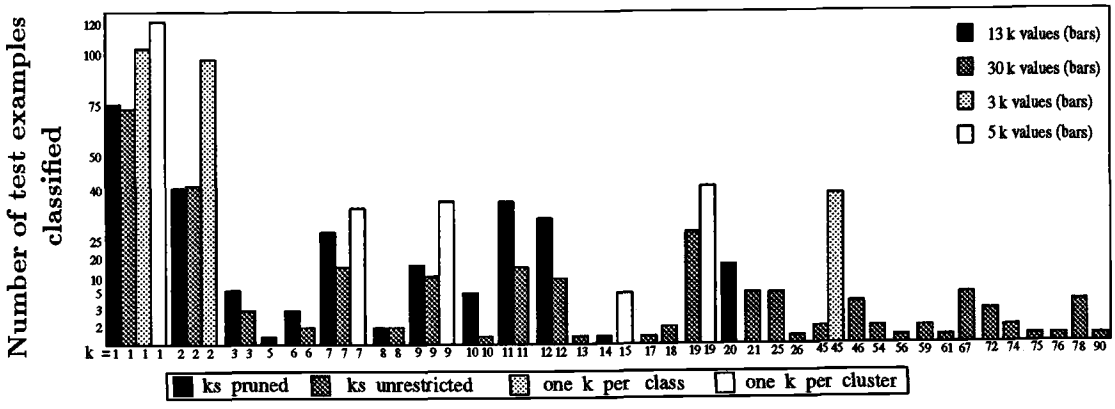


Figure 4.9. Bars show number of times local kNN methods used certain k values to classify test examples from the Sine-Wave data set (Figure 4.5 (Combined), numbers are based on single run). KNN used $k = 1$ in this experiment.

Figure 4.9 shows, for one single run of the Sine-Wave data set (see also Figure 4.5), which k values were actually used by the different local methods. Three clusters of k values can be seen in this graph, one cluster at $k = 1$, one at $k =$ 7,9,11,12 and the third at $k =$ 19,20,21. It is interesting to note that the second and third cluster correspond to the k values used by kNN for the Sine and Waveform-21 data sets, respectively. Furthermore, kNN did not use $k = 1$ in any of the experiments conducted with the Sine or Waveform-21 data sets before these data sets were combined. This gives insight into why kNN's performance was inferior to that of the local methods in this experiment: Patterns in the combined data set belong to one of three categories as indicated by the k values used to classify them ($k = 1$, $k \approx 10$, $k \approx 20$). Hence, the performance difference is due to the fact that kNN must decide at training time which single "k-value" category will give the best performance, while

the local methods make that decision at classification time, separately for each query depending on its local neighborhood.

We can conclude from these experiments that local k-nearest neighbor methods may achieve classification accuracies significantly superior to kNN's on specific data sets. Experiments with commonly used data sets, however, showed no significant differences in performance, in most cases. The most obvious explanation for this behavior is that data sets which are commonly used to evaluate machine learning algorithms may all be similar in that attributes such as distribution of noise or irrelevant features are uniformly distributed across all patterns. In other words, patterns from data sets describing a certain task generally exhibit similar properties.

Local nearest neighbor methods are comparable in computational complexity and accuracy to the (global) k-nearest neighbor algorithm and are easy to implement. In specific applications they can significantly outperform kNN. These applications may be combinations of significantly different subsets of data or may be obtained from physical measurements where the accuracy of measurement depends on the size of the value. Furthermore, local kNN classifiers can be constructed at classification time (on-line learning) thereby eliminating the need for a global cross-validation run to determine the proper value of k.

Two methods can be recommended for domains where attributes such as noise or relevance of attributes vary significantly within different parts of the input space. The first method, called $\text{localKNN}_{ks\ pruned}$, computes a list of "good" k values for each training pattern, prunes less frequent values from these lists, and classifies a query according to the list of k values of a pre-specified number of neighbors of the query. Leave-one-out cross-validation is employed to estimate the proper amount of pruning and the size of the neighborhood that should be used.

The other method, $\text{localKNN}_{one\ k\ per\ cluster}$, applies a cluster algorithm to determine clusters of input patterns. One k is then computed for each cluster and used to classify queries that fall into this cluster. $\text{LocalKNN}_{one\ k\ per\ cluster}$ performs indistinguishably from kNN in all commonly used data sets and outperforms kNN on the constructed data sets. This method compared with all other local methods

discussed in this section introduces a lower computational overhead at classification time, and it is the only method that could be modified to eliminate the need for leave-one-out cross-validation.

The only purely local method, localKNN$_{ks\ unrestricted}$, performs well on constructed data sets and is comparable to kNN on non-constructed data sets. Sensitivity studies (data not shown) indicated that a constant value of 25 for the parameter M gave results comparable to those where cross-validation was employed to determine the value of M. The advantage of localKNN$_{ks\ unrestricted}$ over the other local methods and kNN is that this method does not require any global information whatsoever (if a constant value for M is used). It is therefore possible to construct a localKNN$_{ks\ unrestricted}$ classifier entirely at query-time, which makes this method an attractive alternative for on-line learning or extremely large data sets.

If the researcher has reason to believe that the data set under study is a collection of subsets with significantly varying attributes such as noise or number of irrelevant features, we recommend the construction of a classifier from the training data using localKNN$_{one\ k\ per\ cluster}$ and comparison of its performance to kNN. If the classifier must be constructed on-line, then localKNN$_{ks\ unrestricted}$ should be used instead of kNN.

We conclude that there is considerable evidence that local nearest neighbor methods may significantly outperform the k-nearest neighbor method on specific data sets. We hypothesize that local methods will become relevant in the future when classifiers are constructed that simultaneously solve a variety of tasks.

Table 4.4. Average number of distinct values for k used by local kNN methods.

Task	kNN	local kNN methods			
		ks pruned	*ks unrestricted*	*one k per class*	*one k per cluster*
Letter Recognition	1	7.6±1.1	10.8±1.5	6.4±0.3	1.8±0.2
Led-16	1	16.4±2.5	43.3±0.9	9.2±0.1	9.2±0.5
Combined$_{LL}$	1	52.0±3.8	71.4±1.2	14.7±0.4	3.0±0.2
Sine-21	1	6.6±1.0	27.5±1.1	2.0±0.0	1.0±0.0
Waveform-21	1	9.1±1.4	28.0±1.5	2.9±0.1	4.2±0.2
Combined$_{SW}$	1	13.5±1.5	30.8±1.6	3.0±0.0	4.8±0.2
Constructed	1	11.8±0.9	15.7±0.5	2.0±0.0	5.4±0.2
Iris	1	1.6±0.2	2.0±0.2	2.4±0.1	2.3±0.1
Glass	1	7.7±0.8	11.2±0.7	3.3±0.2	1.9±0.2
Wine	1	2.2±0.4	3.8±0.4	2.0±0.1	2.6±0.1
Hungarian	1	4.1±0.6	12.6±0.6	2.0±0.0	1.0±0.0
Cleveland	1	8.0±1.0	17.2±1.1	1.8±0.1	4.6±0.2
Voting	1	4.1±0.4	6.4±0.3	2.0±0.0	1.3±0.1
Led-7 Display	1	5.6±0.4	7.6±0.4	6.1±0.2	1.0±0.0
Led-24 Display	1	16.0±2.9	37.4±1.6	9.0±0.2	1.6±0.2
Waveform-21	1	9.7±1.3	27.8±1.2	3.0±0.0	4.3±0.1
Waveform-40	1	8.4±2.0	29.9±1.5	3.0±0.0	4.8±0.1
Isolet Letter	1	11.5±2.1	43.9±0.6	16.5±0.5	7.1±0.3
Letter Recognition	1	9.4±1.9	17.0±2.3	6.0±0.3	2.4±0.2

4.2 Should Votes of Neighbors be Weighted by their Distance?

It has been argued that if $k > 1$, then neighbors further away from a query should have less influence on the query's classification. Theoretical results show that in the limit, no weighted vote k-nearest neighbor algorithm (kNN_{wv}, Section 2.1.3) can outperform the basic kNN algorithm [BJ78]. However, it has been argued that in the finite sample case kNN_{wv} may be superior to kNN [MLT87, Wol89, AKA91]. The experiments discussed in this section were designed to determine the conditions under which kNN (with simple majority voting) and kNN_{wv} show different performance.

The performance of kNN and kNN_{wv} is statistically indistinguishable in 13 of the 18 domains in Table 4.5. It is noticeable, however, that kNN_{wv} always performed better than kNN when the average value of k employed by kNN (as determined during cross-validation) is smaller than approximately 15. This better performance of kNN_{wv} is statistically significant in 4 domains. The kNN algorithm gave a superior performance only in the *gaussian* task.

The better performance of kNN_{wv} for small values of k is caused by the fact that kNN_{wv} can minimize the risk of misclassifying the query by using a slightly larger number of neighbors than kNN to classify the query. For example, the average number of neighbors used by kNN in the *banded* task is 1.6 while kNN_{wv} uses 5.4 neighbors. A similar increase in the number of neighbors would result in over-smoothing for kNN, since the additional, more distant, neighbors would have the same influence on the decision as the nearer neighbors (see also Figure 4.10).

The better performance of kNN in the *gaussian* task is due to just the opposite effect: high weight on the more distant neighbors is required for good performance. In this task, the performance of kNN_{wv} was nearly identical for all values of $k > 5$ while the performance of kNN further improved for $k > 25$ (Figure 4.10).

The kNN and kNN_{wv} algorithms assign identical labels to 94.1% of the test cases in the Isolet domain when leave-one-out cross-validation is used to determine the value of k (Figure 4.11). Approximately 3.2 percentage points of the cases where kNN and kNN_{wv} disagree, must be attributed to the fact that kNN and kNN_{wv}

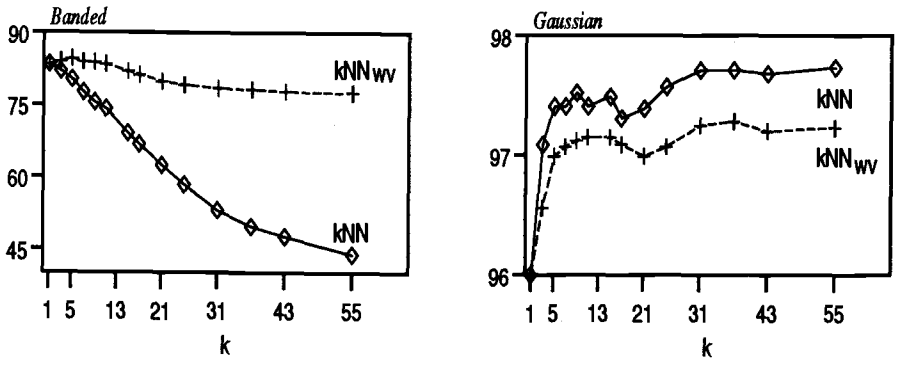


Figure 4.10. Performance of kNN and kNN_{wv} for different values of k in the *banded* and *gaussian* tasks.

often use different k values to classify queries. Although the means of the k values employed by kNN and kNN_{wv} are equal ($\bar{x} = 10.3$, Table 4.5), these algorithms may use different values for k in the different repetitions of the experiment. However, the performance of kNN_{wv} was also superior to that of kNN for all fixed values of $k > 1$. In the case of $k = 5$, the kNN algorithm gave an accuracy of 83.9±0.3% correct, while kNN_{wv} classified 85.3±0.3% of the test examples correctly. When k is fixed at $k = 5$, kNN and kNN_{wv} disagree on only 2.7% of the test cases (Figure 4.12). All of these disagreements can be attributed to the procedure employed by kNN to break ties when several classes are equally frequent among the neighbors of a query. The version of kNN tested in this section breaks these ties quasi randomly. That is, in favor of the class with the smaller number. If this procedure is changed such that kNN uses kNN_{wv} to classify the query whenever there are several classes most frequent among its neighbors, then kNN_{wv} and the amended version of kNN agree on all test cases in the Isolet domain (with $k = 5$).

These experiments strongly support the conclusion that the votes of all k nearest neighbors should have weights inversely proportional to their distances from the query.

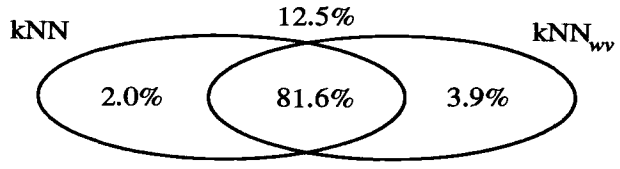


Figure 4.11. Correlation of errors of kNN and kNN$_{wv}$ in the Isolet domain. The value of k was chosen via leave-one-out cross-validation. Percentages inside circles indicate percent correct.

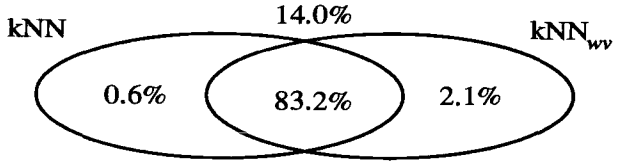


Figure 4.12. Correlation of errors of kNN and kNN$_{wv}$ in the Isolet domain ($k = 5$). Percentages inside circles indicate percent correct.

Table 4.5. The performance of kNN and kNN_{wv} in 18 domains. These (*) differences between kNN and kNN_{wv} are statistically significant ($p < 0.05$). Numbers in parentheses indicate average value of k used by each of the two methods.

Domain	kNN		kNN_{wv}	
	Perf.	avg. k	Perf.	avg. k
Quadrants	97.0±0.4	(10.2±4.1)	97.6±0.3	(32.4±5.8)
Diagonal	97.5±0.3	(6.0±1.9)	97.8±0.3	(20.3±3.2)
Banded	83.0±0.7	(1.6±0.2)	84.1±0.7 *	(5.4±0.5)
Sinusoidal	73.2±0.7	(2.4±1.4)	73.4±0.7	(2.2±1.2)
Radial	86.9±0.6	(1.5±0.2)	87.4±0.6	(6.6±0.8)
Gaussian	97.4±0.2	(26.7±3.3)	97.0±0.3 *	(26.0±3.2)
Iris	95.6±0.4	(6.9±0.9)	95.6±0.4	(12.4±2.1)
Wine	96.2±0.5	(14.8±2.0)	96.8±0.6	(14.8±1.9)
Glass	65.1±0.9	(3.2±1.0)	66.3±1.0 *	(5.6±1.4)
Cleveland	83.1±0.6	(18.5±3.5)	82.9±0.5	(24.1±4.1)
Voting	92.2±0.4	(5.8±0.5)	92.4±0.4	(5.7±0.5)
Hungarian	82.1±1.0	(37.6±2.4)	82.5±1.0	(36.4±3.7)
Waveform-21	81.9±0.8	(30.2±3.1)	81.7±0.8	(30.9±3.7)
Waveform-40	80.8±1.0	(42.2±3.6)	81.3±1.1	(43.1±3.4)
Led-7 Display	72.4±0.6	(4.3±0.4)	73.0±0.6 *	(20.4±2.2)
Led-24 Display	69.5±0.6	(72.2±4.7)	70.2±0.6	(64.5±2.3)
Letter Recognition_{BR}	97.5±0.1	(4.1±0.5)	97.6±0.1	(4.0±0.5)
Isolet	83.6±0.2	(10.3±1.2)	85.5±0.3 *	(10.3±1.0)

4.3 Choosing the Distance Function for k-Nearest Neighbor Algorithms

The experiments described in Section 3.2 have shown that the performance of nearest neighbor algorithms may improve significantly if the distance metric employed to compute the neighbors is well chosen. In contrast to some other learning algorithms such as the multilayer perceptron, the nearest neighbor algorithm with standard Euclidean distance has no means of assigning lower weights to redundant, irrelevant, highly correlated, or very noisy input features. This shortcoming is at least partly to blame for the inferior performance of kNN that is often reported in the machine learning literature. In many cases, the performance reported for kNN can be significantly improved by modifying the distance metric.

For example, Wettschereck & Dietterich [WD92] report in the NETtalk domain an error rate for radial basis function (RBF) networks that is nearly 20 percentage points lower than that of first nearest neighbor (GRBF: 26.2% incorrect, 1-NN: 46.9% incorrect). That performance difference can be nearly eliminated by attaching proper weights[10] to each of the input features in this domain (1-NN with feature weights: 29.1% incorrect). This example illustrates the need for a general method for choosing feature weights for nearest neighbor algorithms.

An issue closely related to the metric is the effect of high input dimensionality on nearest neighbor algorithms. One effect of the "curse of dimensionality" [Bel61, Hub85] is that feature space in high dimensional tasks is mostly devoid of training examples. As a consequence, a given query may not have any exemplars very close to it, but it may have a large number of exemplars that have similar, large, distances to it. Hence, it is conceivable that a reduction in the number of input dimensions could lead to improved performance for nearest neighbor algorithms.

Only very few methods addressing these issues have been proposed in the literature. Dasarathy summarizes some of the results that have been published in the pattern recognition literature [Das91, Chapter 9].

---

[10]How these weights can be computed will be explained in Section 4.3.2

In this section, we will first compare the performance of kNN when distances are computed via the Euclidean L^2-norm versus the performance obtained when the L^1-norm is employed. The main contribution of this section is the development and evaluation of two methods that can be applied to continuous as well as symbolic features to learn the weights of these features. Finally, in Section 4.3.4, a method that can be used to de-correlate or remove continuous features is presented.

Table 4.6. The performance of the nearest neighbor and k-nearest neighbor algorithms in respect to the norm used. Shown is the absolute performance when the L^2-norm is used and the relative difference between the performance obtained with the L^1-norm and that obtained with the L^2-norm. These differences (*) are statistically significant ($p < 0.05$).

Domain	NN		kNN	
	L^2-norm	L^1-norm	L^2-norm	L^1-norm
Quadrants	97.7±0.4	−0.4 *	96.8±0.4	−0.4
Diagonal	97.9±0.3	−0.1	97.2±0.5	−0.2
Banded	83.5±0.7	−0.5	83.0±0.7	−0.3
Sinusoidal	73.6±0.6	−0.7 *	73.2±0.7	−0.6 *
Radial	87.0±0.6	−0.7	86.9±0.6	−0.4
Gaussian	96.0±0.3	+0.1	97.5±0.2	+0.1
Iris	95.2±0.4	−1.2 *	95.6±0.5	−0.7
Cleveland	77.8±0.9	+0.0	83.4±0.5	+0.0
Hungarian	75.9±0.8	+0.0	82.1±1.0	+0.3
Waveform-21	75.2±1.1	+0.1	81.8±0.9	−0.2
Waveform-40	69.4±1.0	+2.0	80.7±1.1	+0.4
Letter Recognition$_{BR}$	97.7±0.1	+0.1	97.5±0.1	−0.1

4.3.1 L^1-Norm versus L^2-Norm

The Euclidean or L^2-norm is the norm most commonly applied to compute distances for nearest neighbor algorithms. The L^2-norm is more heavily influenced by large dif-

ferences in a small number of dimensions than by small differences in a large number of dimensions. This may cause inferior performance in domains with many irrelevant features or with an extremely large number of input features. An alternative norm would be the L^1-norm (see also Section 2.4.5). Results indicate that the performance of NN and kNN generally suffers slightly if the L^1-norm is used instead of the L^2-norm (Table 4.6). The only improvement for both algorithms was observed in the Waveform-40 domain. This gives weak evidence to the hypothesis that the L^1 norm may give superior results in the presence of many irrelevant features.

4.3.2 Mutual Information Feature Weights

The mutual information procedure, as explained in Section 2.4.4, can be applied to estimate the amount of information that each separate input feature provides about the classification labels. This method can be applied without pre-processing to symbolic and nominal features. For continuous features, the probability function of these features must be estimated from the training data through a density estimation technique [Sil86] (see also Section 2.4.4). A feature that perfectly determines the class of its corresponding output has mutual information proportional to the log of the number of classes, while random features have mutual information close to zero.

When used as the weights in a weighted Euclidean distance metric for first nearest neighbor (NN FW_{MI}) and k-nearest neighbor (kNN FW_{MI}), the mutual information values improved NN's and kNN's accuracy by a statistically significant amount in 9 out of 17 domains (Table 4.7). The largest improvements in performance for NN and kNN were observed in domains with many irrelevant features. The decrease in performance in the Led-7 Display domain was probably due to the fact that features in this domain are highly correlated, which violates one of the assumptions made by the mutual information procedure.

Table 4.7. The performance of the nearest neighbor and k-nearest neighbor algorithms with and without feature weights. Feature weights were computed via the mutual information procedure. These differences (*) between the performance without feature weights (NN, kNN) and the performance with feature weights (NN FW_{MI}, kNN FW_{MI}) are statistically significant ($p < 0.05$).

Domain	NN	NN FW_{MI}	kNN	kNN FW_{MI}
Quadrants	97.7±0.4	97.6±0.4	97.0±0.4	97.0±0.4
Diagonal	97.9±0.3	97.9±0.3	97.5±0.3	97.7±0.3
Banded	83.5±0.7	89.2±0.5 *	83.0±0.7	88.8±0.6 *
Sinusoidal	73.6±0.6	75.1±0.7 *	73.2±0.7	74.6±0.9 *
Radial	87.0±0.6	86.7±0.6	86.9±0.6	86.6±0.6
Gaussian	96.0±0.3	96.0±0.3	97.4±0.2	97.9±0.1
Iris	95.2±0.4	95.5±0.4	95.6±0.4	95.6±0.4
Cleveland	77.8±0.9	79.9±0.9 *	83.1±0.6	83.2±0.5
Hungarian	75.9±0.8	77.0±0.7	82.1±1.0	83.9±0.8 *
Voting	86.9±0.8	88.1±0.8 *	92.2±0.4	94.6±0.4 *
Waveform-21	75.2±1.1	76.5±0.9	81.8±0.9	81.8±1.0
Waveform-40	69.4±1.0	73.8±1.0 *	80.8±1.0	83.3±0.8 *
Led-7 Display	70.5±0.6	68.3±0.5 *	72.4±0.6	70.1±0.6 *
Led-24 Display	48.5±0.7	63.1±0.6 *	69.5±0.6	73.2±0.7 *
Letter Recognition_{BR}	97.7±0.1	98.3±0.1 *	97.5±0.1	98.2±0.1 *
Isolet	83.1±0.3	84.4±0.3 *	83.6±0.2	85.1±0.2 *
NETtalk†	55.9	70.9 *	55.9	70.9 *

† Numbers are based on a single experiment.

4.3.3 A Weighted Nearest Neighbor Algorithm with Learned Feature Weights

Lowe [Low94] suggested a method for learning the distance metric to improve the classification accuracy of nearest neighbor algorithms. Based on his ideas, a nearest neighbor algorithm identical to kNN_{wv} (Section 4.2) with a weighted distance metric is introduced in this section. Feature weights are learned by this algorithm via a gradient descent method. The algorithm will be denoted by kNN_{wv} FW_{VSM} to indicate that the votes of the neighbors are weighted inversely proportional to their distance from the query during classification and that it uses a version of Lowe's *variable kernel similarity metric* [Low94] to learn the weights of input features.

Lowe [Low94]'s VSM algorithm (see also Section 2.1.5) passes the distances of the k nearest neighbors of a query through a Gaussian that is centered at the query point. This *activation* of each of the k nearest neighbors determines the weight of the vote of the neighbor when the query is classified. One of the advantages of passing the distances of the neighbors through a Gaussian is that one can then compute the derivative of the error function with respect to the distance metric used. However, any function that properly combines the votes of the nearest neighbors and possesses a derivative could be used instead of the Gaussian. The k-nearest neighbor algorithm, where the votes of the nearest neighbors have weights inversely proportional to their distances to the query, is such an algorithm. This section describes the kNN_{wv} FW_{VSM} algorithm and the experiments that were designed to determine whether Lowe's VSM could be successfully applied to kNN_{wv}.

Computation of the Gradient of kNN_{wv} The gradient with respect to the feature weights can be computed as follows: Let v_{ih} be the i^{th} output of the h^{th} neighbor of q, then the i^{th} output of q, denoted by p_i, is computed as follows:

$$p_i = \frac{\sum_{h=1}^{k} v_{ih}/d(q,u_h)^2}{\sum_{h=1}^{k} 1/d(q,u_h)^2}$$

where:

$$d(q,u)^2 = \sum_{l=1}^{n} w_l^2 (q_l - u_l)^2$$

Define the output error E as the squared sum over all exemplars j and all outputs i of the difference between the desired output (v_{ij}) and the computed output (p_{ij}):

$$E = \sum_{j=1}^{L} \sum_{i=1}^{m} (v_{ij} - p_{ij})^2$$

Then the derivative of E with respect to feature weight w_l can be computed as follows:

$$\begin{aligned}
\frac{\partial E}{\partial w_l} &= -2 \textstyle\sum_{j=1}^{L} \sum_{i=1}^{m} (v_{ij} - p_{ij}) \frac{\partial p_{ij}}{\partial w_l} \\
\frac{\partial p_{ij}}{\partial w_l} &= \frac{\sum_{h=1}^{k} (v_{ih} - p_{ij}) \partial d(q,u_h)^{-2} / \partial w_l}{\sum_{h=1}^{k} d(q,u_h)^{-2}} \\
\frac{\partial d(q,u_h)^{-2}}{\partial w_l} &= -1 \times d(q,u_h)^{-4} \times \frac{\partial (\sum_{l=1}^{n} w_l^2 (q_l - u_l)^2)}{\partial w_l} \\
\frac{\partial d(q,u_h)^{-2}}{\partial w_l} &= -2 \times d(q,u_h)^{-4} \times w_l (q_l - u_{lh})^2 \\
\frac{\partial d(q,u_h)^{-2}}{\partial w_l} &= -2 \times \frac{w_l (q_l - u_{lh})^2}{d(q,u_h)^4}
\end{aligned}$$

Experiments with kNN$_{wv}$ FW$_{VSM}$ The experiments described in Section 3.2 indicate that kNN is more sensitive to the proper choice of the distance function than first nearest neighbor. Hence, a distance metric that leads to good results for $k > 1$ should, in general, also give good results for $k = 1$. However, when k is chosen too large, then the error (and therefore the gradient) computed may be substantially different from the error that would be computed for the optimal k. Initial experiments showed that good results can be obtained in most domains with $k = 25$. A general method for determination of the optimal number of neighbors that are necessary to compute the error and gradient while feature weights are learned ("vsm-training") should be the topic of future research.

A modified version of the conjugate gradient module by Barnard and Cole [BC89] was used in this research to adjust the feature weights during vsm-training. The number of training epochs was limited to 10. For each training example, the first 100 nearest neighbors were initially determined and stored in a priority queue according to their distances from the example. Only the distances of the first k neighbors

of the example were re-computed during each epoch while all 100 distances were re-computed whenever the conjugate gradient procedure started a new line search.

Leave-one-out cross-validation was used after feature weights were learned to estimate the optimal value of k.

Results indicate that this procedure is very effective in estimating good feature weights (Table 4.8). Particularly the results obtained in the *banded* and *sinusoidal* tasks are substantially better than those obtained with the mutual information procedure. The vsm-training procedure currently employed appears to be unable to learn the proper feature weights in the Waveform domains. This deficiency may be due to the fixed value of k used during vsm-training. The procedure also appears to be inferior to the mutual information procedure in domains with symbolic features (Cleveland and Hungarian domains). Nonetheless, the performance of kNN_{wv} FW_{VSM} was never significantly worse than kNN's. It was significantly superior to kNN in 7 out of 18 domains. The kNN_{wv} FW_{VSM} algorithm performed significantly better than kNN_{wv} FW_{MI} in 2 domains, while kNN_{wv} FW_{MI} was superior in 2 other domains. These results suggest that the VSM feature weight learning procedure is a reliable procedure for learning feature weights in a general setting and that it should be employed in all domains with continuous features and in domains with symbolic features if there is evidence that features are highly correlated.

4.3.4 De-correlation and Removal of Features via Principal Component Analysis

Input features that are highly correlated can cause distance-based algorithms to compute distances that may not reflect the optimal distance between two data points. For example, two input features may be identical. The effect of these two identical input features is equivalent to a single feature with twice the weight during distance calculations. The feature's larger weight is only justified if it contains more information with respect to the desired outputs than the other features. Otherwise the larger weight will result in a degradation in classification accuracy. De-correlation of input features may therefore improve the classification accuracy of distance-based

Table 4.8. The performance of the weighted vote kNN algorithm without feature weights (kNN_{wv}), with computed feature weights (kNN_{wv} FW_{MI}), or learned feature weights (kNN_{wv} FW_{VSM}).

Domain	kNN_{wv}	kNN_{wv} FW_{MI}	kNN_{wv} FW_{VSM}
Quadrants	97.6±0.3	97.4±0.4	97.3±0.4
Diagonal	97.8±0.3	97.7±0.3	97.8±0.4
Banded	84.1±0.7	89.5±0.7 *	95.7±0.3 *
Sinusoidal	73.4±0.7	74.6±0.7 *	83.5±0.8 *
Radial	87.4±0.6	87.4±0.6	87.3±0.7
Gaussian	97.0±0.3	97.1±0.3	97.1±0.2
Iris	95.6±0.4	95.6±0.4	95.0±0.6
Cleveland	82.9±0.5	83.7±0.6	82.7±0.7
Hungarian	82.5±1.0	83.9±0.8 *	81.6±1.0
Voting	92.4±0.4	94.7±0.4 *	95.0±0.4 *
Waveform-21	81.7±0.8	82.2±0.9	81.9±0.9
Waveform-40	81.3±1.1	83.4±0.8 *	81.0±1.1
Led-7 Display	73.0±0.6	71.3±0.6 *	72.8±0.6
Led-24 Display	70.2±0.6	73.5±0.6 *	72.9±0.6 *
Letter Recognition_{BR}	97.6±0.1	98.2±0.1 *	98.2±0.1 *
Isolet	85.5±0.3	86.1±0.3 *	86.1±0.3 *
NETtalk	55.3	71.8 *	70.4 *

algorithms and specifically of nearest neighbor algorithms. The method of principal component analysis (PCA, Section 2.4.3) can be used to transform continuous input features into features that are mutually independent from each other. Henceforth, the new features resulting from principal component analysis will be referred to as "pca-features". Pca-features are mutually de-correlated and they generally have substantially different variances (as indicated by each pca-feature's eigenvalue).

In this section, we will investigate the issue of whether principal component analysis can be successfully applied to classification tasks. Two conditions must be

satisfied for this method to succeed in supervised learning tasks: (1) The pca-features with the largest eigenvalues must also carry the most information with respect to the classification. (2) The linear transformation applied to the data must not overshadow any structure that was present in the data before PCA was applied. Such structure may assist the algorithm in learning the task.

Four methods for incorporating pca-features into the k-nearest neighbor algorithm are compared in this section.

1. Each pca-feature can be normalized into the range [0...1]. This is the standard method as used throughout this research.

2. Each pca-feature can be normalized and then weighted by its eigenvalue. Each pca-feature's eigenvalue indicates the amount of "structure" from the original data that was transferred into this pca-feature. The eigenvalue may thus be a good indicator of the feature's relevance.

3. Pca-features can be left un-normalized.

4. Pca-features can be left un-normalized and be weighted by their eigenvalues.

Method 1. can only be recommended in domains that have symbolic as well as continuous features (Figure 4.13). In such domains, un-normalized (or weighted) pca-features would dominate all distance calculations due to the fact that the largest eigenvalue is generally very large. For example, the largest eigenvalue in the Cleveland and Hungarian domains is generally larger than 2000.

When kNN was trained on the normalized pca-features in domains that have only continuous features, performance was never inferior to kNN trained on the original features (Figure 4.13). In the Iris domain, a significant improvement in predictive accuracy was obtained when method 2. was applied to the pca-features (Figure 4.13). In both Waveform domains, a significant improvement in performance was observed for methods 2. and 4., as well as for method 3. in the Waveform-40 domain (Figures 4.13 and 4.14). In the Letter Recognition$_{BR}$ domain, performance was significantly inferior to kNN trained on the original features when pca-features

were not normalized. This inferior performance may be due to the fact that features in the Letter Recognition domain are nominal and not continuous. These results indicate that PCA can be applied to the features of tasks with solely continuous features, and that pca-features should not be normalized (method 3.). This procedure may lead to a significant improvement in performance in domains with irrelevant features.

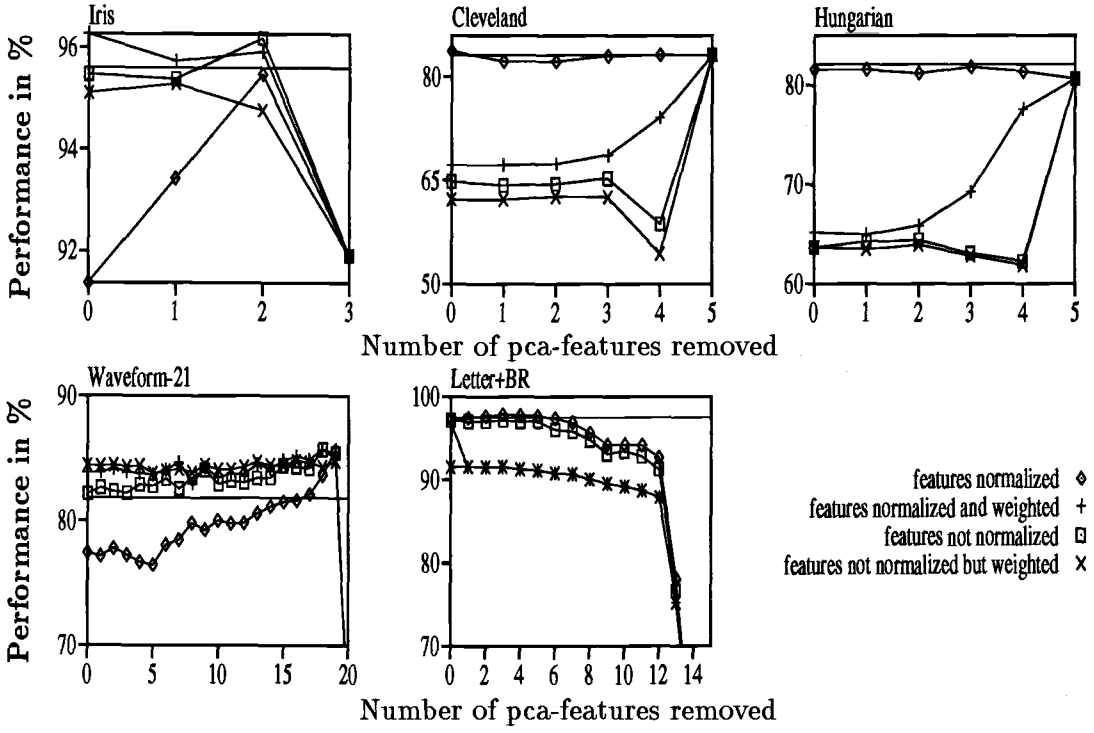


Figure 4.13. Change in classification accuracy as an increasing number of features that have been transformed via PCA are removed from non-synthetic data sets. The four lines in each graph show the performance of kNN when features are (not) normalized and/or weighted by their eigenvalues. In the Cleveland and Hungarian domains, PCA was only applied to the 5 continuous features. The solid line in each graph indicates the performance of kNN when trained and tested on the original features.

The fact that pca-features should not be normalized for best performance indicates that those pca-features with the largest eigenvalues do indeed carry the largest amount of information with respect to the classification. This implies that it may be possible to remove some of the pca-features with low eigenvalues. For example, all but the 2 pca-features with the largest eigenvalues were removed in the Wave-

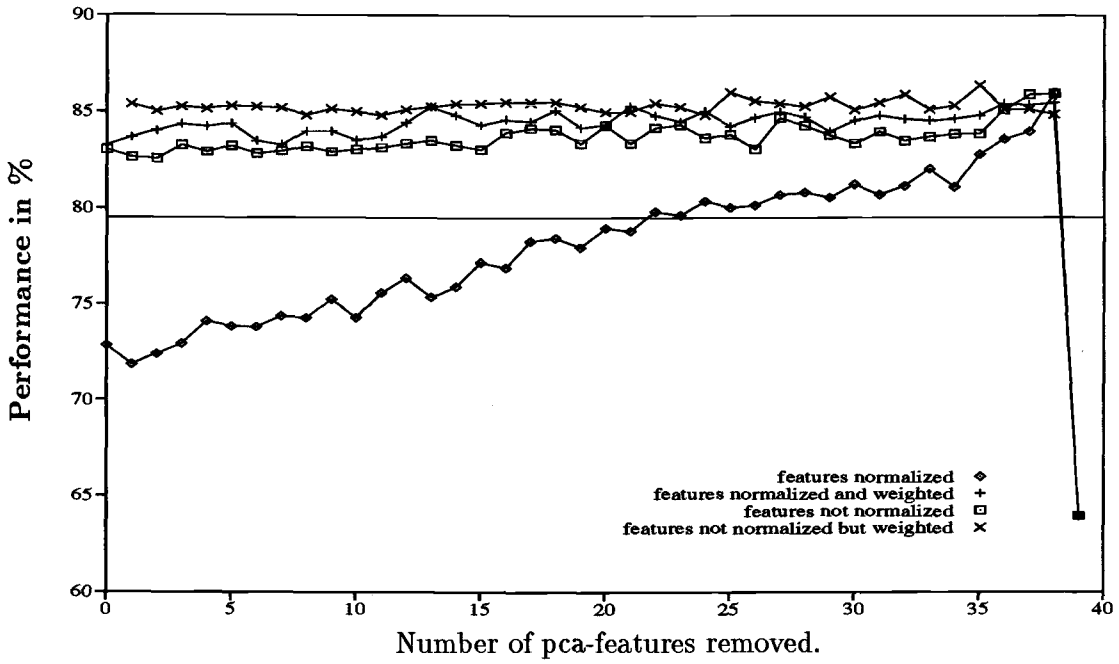


Figure 4.14. Accuracy of kNN in Waveform-40 domain when an increasing number of features with the lowest eigenvalues is removed after initial features have been transformed with PCA.

form domains without any loss in predictive accuracy (Figures 4.13, 4.14, and 4.15). Through removal of 38 pca-features a significant gain in accuracy as compared to the performance of kNN on the original data was obtained. Best performance was obtained in the Iris domain when 2 of the 4 pca-features were removed (Figure 4.13). Further, 5 out of 16 pca-features in the Letter Recognition domain (trained and tested on letters B and R, only) were removed without any significant loss in predictive accuracy. A surprising result in the Cleveland and Hungarian domains was that all continuous features could be removed in these domains without any significant loss in predictive accuracy. In a selected experiment in the Isolet domain, the same predictive accuracy was achieved with 117 of the 617 pca-features (not normalized) as with all original features.

These results strongly support the conclusion that principal component analysis should be used as a preprocessing method for distance-based algorithms. This method can be successfully applied to remove irrelevant features in order to improve the performance and speed of the final classifier. In an attempt to develop a general

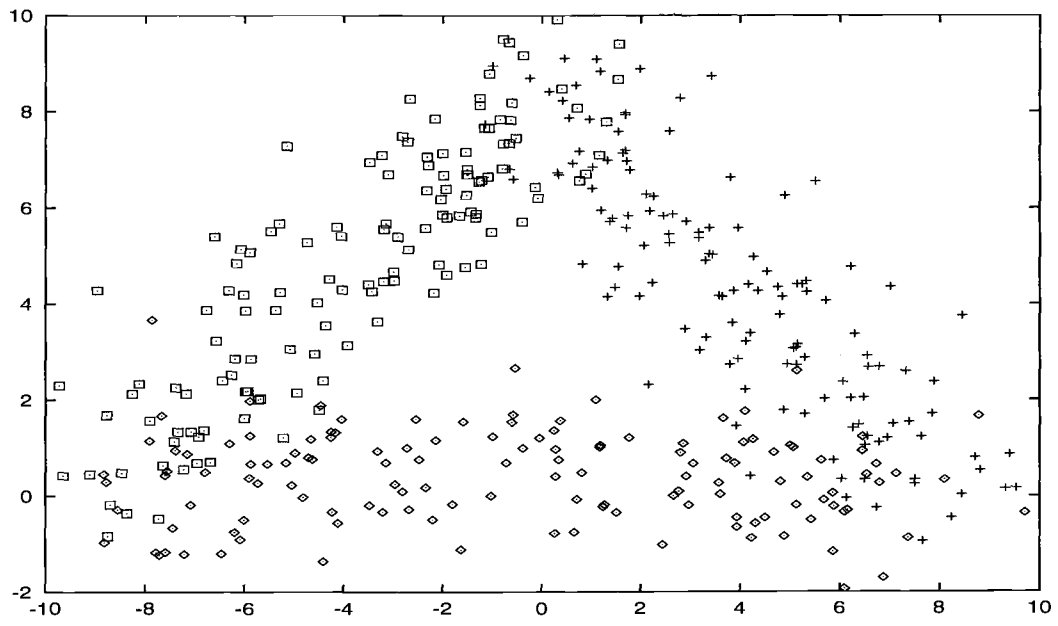


Figure 4.15. Location of training examples of the Waveform-40 domain as indicated by the two features with the largest eigenvalues after PCA has been applied to initial features.

method for determining the number of pca-features that should be removed for best performance, we compared the eigenvalues computed in the Waveform-40 and Letter Recognition domains. The eigenvalues of approximately 30 of the pca-features that were removed in the Waveform-40 domain were larger than the eigenvalue of the pca-feature with the smallest eigenvalue in the Letter Recognition that had to be retained for best performance. Similarly, ratios of eigenvalues also failed to provide any general information. In view of these results, further research is necessary to obtain a general method to determine the proper number of pca-features that should be removed for best performance.

Principal components analysis has two main disadvantages. Firstly, its demand on computational resources is high when the number of dimensions is high. Secondly, the transformed features have no apparent relation to the original features and can, thus, often not be assigned a meaning.

4.3.5 Combining PCA and Feature Weight Learning Methods

One significant assumption made in Section 4.3.2 is that features are independent from each other. Hence, mutual information can be computed for each feature independently from all other features. Naturally, this may not always be true. However, we have already seen in Section 4.3.4 that input features can be de-correlated without any significant loss of information via the principal component analysis procedure. The principal component analysis procedure also decreased the number of relevant input features in all of the experiments described in Section 4.3.4. The two methods for determining the weights of input features discussed in the previous sections substantially improved the performance of kNN in data sets with irrelevant features. Hence, it is conceivable that these feature weighting methods may substantially improve the performance of kNN trained in pca-features.

The following experiments were conducted. Mutual information feature weights were computed for the normalized pca-features and kNN was trained on the pca-features with these feature weights. The performance of kNN FW_{MI} (pca-features) was significantly inferior to the performance of kNN FW_{MI} (original features) in 5 out of 12 domains and significantly superior in only 1. The drop in performance in the Waveform domains is especially disappointing since we know that most of the pca-features in these domains are irrelevant and could have weight 0. It appears that the density estimation technique employed was unable to estimate the proper distribution of the features in these domains. Further research is therefore necessary to obtain satisfactory performance with this technique.

Better results were obtained when kNN_{wv} FW_{VSM} was trained on the un-normalized pca-features. Performance was significantly inferior only in the *banded* and *sinusoidal* tasks and significantly superior in the two Waveform domains and the *diagonal* task. The poor results in the *banded* and *sinusoidal* tasks are not surprising. The data used to generate these tasks were drawn from uniform distributions. Hence, there was no structure whatsoever in the input data. The feature weighting algorithms were able to determine in the *banded* task, for example, that the horizontal dimension of the

original data was irrelevant. However, that information was not preserved when PCA was applied, which explains the poor performance. The relatively good performance of kNN_{wv} FW_{VSM} on the non-synthetic data sets, on the other hand, is encouraging. The results suggest that in most tasks, principal component analysis should preserve or even enhance (as it did in the Waveform domains) the structure that was present in the original data. A good feature weighting algorithm such as FW_{VSM} can then reduce the number of effective input dimensions and thus significantly improve the performance of distance-based algorithms.

Table 4.9. The performance of the k-nearest neighbor algorithms with and without features weights when trained on the original features or the pca-features. Feature weights were computed via the mutual information procedure (L^2_{MI} norm) or learned (L^2_{VSM} norm).

Domain	original features			pca-features		
	kNN		kNN_{wv}	kNN		kNN_{wv}
	no FW	FW_{MI}	FW_{VSM}	no FW	FW_{MI}	FW_{VSM}
Quadrants	97.0±0.4	97.0±0.4	97.3±0.4	97.0±0.4	96.8±0.4	97.6±0.3
Diagonal	97.5±0.3	97.7±0.3	97.8±0.4	97.5±0.3	98.3±0.3 *	98.4±0.2 *
Banded	83.0±0.7	88.8±0.3	95.7±0.3	82.8±0.8	83.0±0.7 *	84.6±0.7 *
Sinusoidal	73.2±0.7	74.6±0.7	83.5±0.8	73.3±0.9	72.6±0.7 *	76.6±0.8 *
Radial	86.9±0.6	86.6±2.1	87.3±0.7	86.8±0.6	86.6±0.6	87.0±0.6
Gaussian	97.4±0.2	97.9±0.2	97.1±0.2	97.3±0.1	97.3±0.3	97.0±0.3
Iris	95.6±0.4	95.6±0.5	95.0±0.6	95.5±0.4	95.2±0.7	95.6±0.6
Cleveland	83.1±0.6	83.2±0.4	82.7±0.7	84.0±0.5	83.2±0.5	82.3±0.7
Hungarian	82.1±1.0	83.9±0.9	81.6±1.0	82.1±1.8	81.9±0.9 *	81.0±0.8
Wave-21	81.8±0.9	81.8±0.9	81.9±0.9	82.3±0.9	81.0±1.0	85.2±1.0 *
Wave-40	80.8±1.0	83.3±0.8	81.0±1.1	83.0±0.8 *	80.9±1.0 *	85.1±0.9 *
Letter $R._{BR}$	97.5±0.1	98.2±0.2	98.2±0.1	97.1±0.1 *	97.6±0.1 *	97.1±0.3 *
Isolet	83.6±0.2	85.1±0.2	86.1±0.3	83.5±0.3		

4.4 Summary

An extensive study of the k-nearest neighbor algorithm has been conducted in this chapter. We have shown that for best performance it is necessary to conduct cross-validation on the value of k. Several methods of cross-validation were compared and it was found that, in general, the best estimate of the optimal value of k is obtained by leave-one-out cross-validation. However, one-fold cross-validation was shown to be a competitive alternative to leave-one-out cross-validation if the time required to determine the value of k is of concern. It was also shown that the performance of kNN is not sensitive to the exact choice of the value of k when k must be large. This, in turn, led to the observation that for best performance it is necessary to conduct cross-validation on a restricted number of values for k only. We suggest to conduct cross-validation for k on these values: 1,3,5,7,9,13,17,27,35,41. The smaller values were chosen, since it was shown that cross-validation on all values of k and cross-validation odd values only leads to statistically indistinguishable results. The larger values were chosen arbitrarily due to kNN's insensitivity to the exact choice of k when k is large.

Four methods for the determination of the value of k from local data were presented in Section 4.1.2. It was shown that local kNN methods can significantly outperform kNN on data sets where noise or relevance of features varies substantially in different parts of the input space.

The kNN algorithm with majority voting (denoted by kNN) was compared to the kNN algorithm where the votes of the neighbors have weights inversely proportional to their distance from the query during computation of the query's classification (denoted by kNN_{wv}). The kNN algorithm was found to perform significantly worse than kNN_{wv} in 4 out of 18 domains, and was superior only in the *gaussian* task. We recommend to employ kNN_{wv} instead of kNN in all domains.

The issue of how to estimate the proper distance metric for nearest neighbor algorithms is investigated in Sections 4.3.2 and 4.3.3. A procedure based on the mutual information between input features and output classes was shown to give results

significantly superior to that of kNN with standard Euclidean distance in 9 out of 17 domains. This procedure is very fast and can be applied directly to symbolic and nominal features. The mutual information procedure requires estimation of the probability distribution of continuous features. The quality of this estimate can greatly affect the quality of the feature weights computed. A second method, called FW_{VSM}, which learns feature weights via gradient descent, was developed. It was shown to significantly improve the performance of kNN_{wv} in 7 out of 17 domains, and never to perform significantly worse than kNN_{wv} without feature weights. Finally, principal component analysis (PCA) is evaluated as a method for reducing the number of relevant dimensions in supervised learning tasks. It was shown that in domains with continuous features, a large number of input features could be removed without any significant loss in predictive accuracy after PCA has been applied. This method, when combined with the FW_{VSM} feature weight learning algorithm, boosted the performance of kNN_{wv} in the Waveform domains from 81% correct to 85% correct which is very close to the Bayes optimal classification rate of 86% correct [BFOS84]. We conclude that either the mutual information or the VSM feature weight learning algorithms should be employed whenever nearest neighbor algorithms are used and that features should be pre-processed via PCA in domains with solely continuous features.

4.5 Related Work

Nearest neighbor algorithms have been the subject of interest in a variety of research areas for many years. An excellent survey of many of the publications relevant to nearest neighbor algorithms has been published by Dasarathy [Das91]. An important area of research discussed by Dasarathy relates to reducing the storage requirements of kNN [Das91, Chapter 6]. Hart's [Har68] *condensed nearest neighbor* (CNN) algorithm only stores training examples if they are misclassified by the previously stored examples. The CNN algorithm is very similar to Kibler and Aha's IB2 [KA87]. Wilson [Wil72] and Tomek [Tom76a, Tom76b] discuss several methods for removing

training examples from the design set. All of these methods generally result in a substantial decrease in the storage requirements of kNN. In some tasks, editing can improve the performance of the classifier [Wil72, Tom76a, Aha90].

Aha [Aha90] investigated the nearest neighbor algorithm in detail from a machine learning perspective. He presents several modifications of the basic algorithm which he terms IB1, IB2, IB3, and IB4, indicating four different instance based machine learning algorithms. He mathematically analyses the simplest algorithm, IB1, and presents theorems proving its convergence. IB1 is identical to kNN_{wv} (Section 4.2). IB2 was designed to reduce the storage requirements of IB1 by storing only those training examples that would be misclassified during incremental learning by the already stored examples. IB2 is rather sensitive to noise in the data, necessitating the development of IB3, a noise-tolerant version of IB2. IB3 maintains a prediction record with each instance, and uses only those instances with good records during prediction. Instances with very poor prediction records are deleted from the set of stored exemplars. IB4 is an extension of the IB3 algorithm that also learns feature weights in a manner similar to that used by Salzberg [Sal91] (see also Chapter 4, Section 5.7). Aha [Aha90] reports substantially superior performance for IB4 over IB1 in the presence of many irrelevant features. However, this superior performance was only obtained when k was kept fixed at $k = 1$ for IB1. This confirms the results from Section 3 where we have shown that an increase in the value of k can be used to compensate for some noisy instances and/or irrelevant features. Hence, in a fair comparison, one would have to compare IB4 with kNN.

An alternative approach to improving the performance and/or reducing the storage requirements of nearest neighbor algorithms was presented by Kohonen [KBC88, Koh89, Koh90b, Koh90a]. Kohonen termed his method *Learning Vector Quantization* (LVQ, see also Section 2.1.6). The LVQ algorithm stores a subset of the training data. The stored exemplars are iteratively moved to different input locations until a termination condition is satisfied. None of the LVQ methods presented in the literature to date learns or computes the weights of the different input dimensions explicitly. However, one could argue that LVQ-type algorithms implicitly

learn the relevance of the different features by adjusting the coordinates of irrelevant input features for all stored exemplars such that they are identical. The result of such an adjustment would be that all stored exemplars have identical distance to all queries at that feature thereby effectively eliminating that feature. This effect is illustrated for Generalized Radial Basis Function networks [PG89] by Wettschereck & Dietterich [WD92].

Stanfill & Waltz's [SD86] present a method for computing meaningful distances between symbolic features. They termed their metric the *Value Difference Metric* (VDM). The Value Difference Metric computes a distance for each pair of the different values a symbolic feature can assume. It essentially compares the relative frequencies of each pair of symbolic values across all classes. Two feature values have a small distance if their relative frequencies are approximately equal for all output classes. Cost & Salzberg [CS93] present a nearest neighbor algorithm that uses a modification of VDM. The main difference between Cost & Salzberg's method and VDM is that their method's feature value differences are symmetric. This is not the case for VDM.

Finally, a data-dependent metric is suggested by Short & Fukunaga [SF80, SF81] and Fukunaga & Flick [FF84] in the case of two class problems and by Myles & Hand [MH90] for the multiclass problem. Fukunaga & Flick [FF82] also present an approach to computing the distance metric if the underlying probability distribution of the data can be estimated.

Chapter 5
An Evaluation of Nearest Hyperrectangular Algorithms

One of the main disadvantages of the algorithms discussed in Chapter 4 is the large amount of memory they require to store the training data. The algorithms discussed in this chapter attempt to find more compact representations of the training data by constructing hyperrectangles that represent a collection of training examples that belong to the same class. More compact representations of the training data lead to faster classification times and may increase the ability of the user to understand decisions made by the classifier.

Salzberg [Sal91] describes a family of learning algorithms based on nested generalized exemplars (NGE). In NGE, an exemplar is a single training example, and a generalized exemplar is an axis-parallel hyperrectangle that may cover several training examples. These hyperrectangles may overlap or nest. The NGE algorithm grows the hyperrectangles incrementally as training examples are processed. Once the generalized exemplars are learned, a test example can be classified by computing the Euclidean distance between the example and each of the generalized exemplars. If an example is contained inside a generalized exemplar, the distance to that generalized exemplar is zero. The class of the nearest generalized exemplar is output as the predicted class of the test example.

The NGE approach can be viewed as a hybrid of nearest neighbor methods (Chapter 4) and propositional Horn clause rules [SS86]. Like nearest neighbor methods, a distance metric is applied to match test examples to training examples. But like Horn clause rules, the training examples can be generalized to be axis-parallel hyperrectangles.

Salzberg [Sal91] reported promising classification results in three domains. However, as reported below, when NGE was tested in 11 additional domains, it gave less

accurate predictions in many domains as compared to the k-nearest neighbor (kNN) algorithm. The goal of this chapter is to demonstrate this performance, understand its causes, and test algorithm modifications that might improve NGE's performance.

The first part of this chapter is devoted to a study that compares NGE and kNN. To equitably compare these algorithms, it was necessary to find optimal settings for various parameters and options in NGE and kNN. Thus, we first describe a series of experiments studying how the performance of NGE is determined by several key parameters and options. These include the number of starting seeds, the treatment of un-generalized exemplars, and the treatment of nominal feature values. We then present results showing that NGE (under the best parameter settings) is substantially inferior to kNN in 9 of the 11 domains tested and superior to kNN in 2 of these domains.

The second part of this chapter attempts to diagnose and repair the causes of this performance deficit. We present several hypotheses including (a) inappropriateness of the nested hyperrectangle bias, (b) inappropriateness of the overlapping of hyperrectangle bias, and (c) poor performance of the search algorithm and heuristics for constructing hyperrectangles. Experiments are then presented that test each of these hypotheses. A version of NGE (called NONGE) that disallows overlapping rectangles while retaining nested rectangles and the same search procedure was uniformly superior to NGE in all 11 domains and significantly better in 6 of them. A batch algorithm (OBNGE) that incorporates an improved search algorithm and disallows nested rectangles (but still permits overlapping rectangles) was only superior to NGE in one domain (and worse in two). These and other experiments lead us to conclude that a major source of problems in NGE was the creation of overlapping rectangles.

We also present a batch version of NONGE, called BNGE, that is very efficient and requires no user tuning of parameters. We recommend that BNGE be employed in domains where batch learning is appropriate. The amount of memory required by NGE algorithms can be further reduced after the classifier is constructed by pruning hyperrectangles that were not generalized during the training period. These trivial

hyperrectangles contribute little to NGE's predictive accuracy, but consume memory and increase the time needed for classification.

The ideal behavior for a hybrid algorithm like NGE would be to take advantage of axis-parallel rectangles where appropriate to find concise, interpretable representations of the learned knowledge. However, in domains where axis-parallel rectangles are not appropriate, NGE would behave more like a nearest neighbor algorithm. The versions of NGE that we developed do take advantage of hyperrectangles, but they perform poorly in domains where hyperrectangles are inappropriate. Hence, the potential advantages of NGE algorithms (data compression, fast learning and classification, interpretability of exemplars) are significant, but classification accuracy is still not satisfactory. An additional modification to the NGE algorithm is proposed and tested in Section 5.6. To achieve better classification accuracy, a hybrid algorithm that uses BNGE in areas of the input space that are covered by hyperrectangles and that uses kNN otherwise, is introduced and evaluated. We call this algorithm KBNGE. While BNGE was significantly inferior to the kNN algorithm in 7 out of 11 domains, KBNGE was inferior to the kNN algorithm in 4 domains and yielded better predictive accuracy in 2 other domains. Furthermore, KBNGE was faster than kNN in all domains. The KBNGE algorithm can be seen as a generalized Nearest Neighbor algorithm. Nearest Neighbor algorithms play an important role in inductive machine learning because of their simplicity and their ability to give highly accurate predictions after a short learning phase (see also Chapter 4). The KBNGE algorithm shares these advantages and offers, in addition, fast classification and a compact representation of those parts of the task learned that clearly belong to a certain class.

The third part of this chapter takes up the issue of learning feature weights for a weighted Euclidean distance. Salzberg [Sal91] proposed an online weight adjustment algorithm. Data are presented showing that this algorithm often performs poorly and erratically. An alternative feature weight algorithm, based on mutual information (c.f. Section 2.4.4), is shown to work well with NN, NGE, and BNGE.

Section 5.8 compares the best version of NGE (BNGE FW_{MI}) and the hybrid algorithm developed in Section 5.6, KBNGE FW_{MI}, to the best version of kNN (kNN_{cv} FW_{MI}). The comparison showed that despite the improvements in NGE, it still was significantly inferior to kNN in 7 domains and significantly superior in 2 domains. When compared to single nearest neighbor (NN), the best version of NGE fared slightly better: it was significantly superior in 3 domains and significantly inferior in 5. The hybrid algorithm with feature weights was significantly inferior to kNN with feature weights in 3 domains and superior in the two rectangular domains while it was superior to first nearest neighbor with feature weights in 7 domains and inferior in only 2.

5.1 The NGE algorithm

Figure 5.1 summarizes the NGE algorithm closely following Salzberg's [Sal91] definition of NGE. NGE constructs hyperrectangles by processing the training examples one at a time. It is initialized by randomly selecting a user-defined number of seed training examples and constructing trivial (point) hyperrectangles for each seed. Each new training example is first classified according to the existing set of hyperrectangles by computing the distance from the example to each hyperrectangle. If the class of the nearest hyperrectangle and the training example coincide, then the nearest hyperrectangle is extended to include the training example, otherwise the second nearest hyperrectangle is tried. (This is called the second match heuristic.) Should both the first and second nearest hyperrectangles have different classes than the training example, then the training example is stored as a new (trivial) hyperrectangle. A query is classified according to the class of the nearest hyperrectangle. Distances are computed as follows: If an example lies outside of all existing hyperrectangles, a weighted Euclidean distance to the nearest side of each hyperrectangle is computed. If the example falls inside a hyperrectangle, its distance to that hyperrectangle is zero. If the example is equidistant to several hyperrectangles, the hyperrectangle that is smallest in area is chosen.

1. **Build an NGE classifier (input: number s of seeds):**
2. Initialization: /* assume training examples are given in random order */
3. for each of the first s training examples E^s call createHyperrectangle(E^s)
4. Training:
5. for each remaining training example E:
6. find the two H^j with $D(E, H^j)$ minimal
7. /* in case of ties, choose the two H^j with minimal area */
8. call these hyperrectangles $H^{closest}$ and $H^{second\ closest}$
9. if (compare($H^{closest}$,E)) generalize($H^{closest}$,E)
10. else if (compare($H^{second\ closest}$,E)) generalize($H^{second\ closest}$,E)
11. else createHyperrectangle(E)

12. **Compare classes of a hyperrectangle and an example:**
13. compare(H, E)
14. if (class(E) == class(H)) return true else return false

15. **Generalize a hyperrectangle:**
16. generalize(H, E)
17. for all features of E do:
18. $H_{upper,f_i} = \max(H_{upper,f_i}, E_{f_i})$
19. $H_{lower,f_i} = \min(H_{lower,f_i}, E_{f_i})$
20. replMissFeatures(H,E)

21. **Create a hyperrectangle:**
22. createHyperrectangle(E)
23. $H_{upper} = E$
24. $H_{lower} = E$
25. $H_{area} = 0$
26. replMissFeatures(H, E)

27. **Replace missing features in a hyperrectangle:**
28. replMissFeatures(H,E)
29. for all features of E do:
30. if (feature i of E is missing)
31. $H_{upper,f_i} = 1$
32. $H_{lower,f_i} = 0$

33. **Classification of a test example:**
34. classify(E)
35. output: class(H^j) with $j = \text{argmin}_i$ D(E,H^i)
36. /* in case of ties, choose H^j out of all ties with minimal area */

Figure 5.1. Pseudo-code describing construction of an NGE classifier and classification of test examples. H generally denotes a hyperrectangle and E an example.

In our implementation of NGE, we first make a pass over the training examples and normalize the values of each feature into the interval [0,1] (linear normalization [Aha90]). Features of values in the test set are normalized by the same scaling factors (but note that they may fall outside the [0,1] range). Aside from this scaling pass, the basic algorithm is entirely incremental.

Each hyperrectangle H^j is labeled with an output class. The hyperrectangle is represented by its lower left corner (H^j_{lower}) and its upper right corner (H^j_{upper}). The distance between H^j and an example E with features f_1 through $f_{nFeatures}$ is defined as follows:

$$D(E, H^j) = w_{H^j} \times \sqrt{\sum_{i=1}^{nFeatures} \left(w_{f_i} \times d_{f_i}(E, H^j)^2\right)}$$

where:

$$d_{f_i}(E, H^j) = \begin{cases} E_{f_i} - H^j_{upper,f_i} & \text{if } E_{f_i} > H^j_{upper,f_i} \\ H^j_{lower,f_i} - E_{f_i} & \text{if } H^j_{lower,f_i} > E_{f_i} \\ 0 & \text{otherwise} \end{cases}$$

w_{f_i} weight of feature i (see Section 5.7)

w_{H^j} weight of hyperrectangle j, computed as the ratio of the number of training examples classified by H^j by the number of training examples correctly classified by H^j:

$$w_{H^j} = \frac{\text{number of times compare}(H^j, E_x) \text{ was called}}{\text{number of times compare}(H^j, E_x) \text{ returned true}}$$

The original NGE algorithm was designed for continuous features only. Discrete and symbolic features require modifications of the distance and area computations for NGE. We adopted the policy that for each symbolic or discrete feature, the set of covered feature values is stored for each hyperrectangle (analogous to storing the range of feature values for continuous features). A hyperrectangle then covers a certain feature value if that value is a member of the covered set. If a hyperrectangle is generalized to include a missing discrete or symbolic feature, then a flag is set such that the corresponding feature of the hyperrectangle will cover any feature value in the future.

The area of non-trivial hyperrectangles is then computed as follows:[11]

$$area(H) = \prod_{i=1}^{nFeatures} size(H_{f_i})$$

with $size(H_{f_i})$ computed as follows:

37. if (H_{f_i} has been generalized to include a missing feature) $size(H_{f_i}) = 1$
38. else if (f_i is continuous)
39. if ($H_{upper,f_i} == H_{lower,f_i}$) $size(H_{f_i}) = 1$
40. else $size(H_{f_i}) = H_{upper,f_i} - H_{lower,f_i}$
41. else /* f_i is a discrete or symbolic feature */
42. $size(H_{f_i}) = \frac{\text{number of values of } f_i \text{ covered by } H}{\text{number of possible values of } f_i}$

Note, therefore, that the maximum possible area of a hyperrectangle is 1. Furthermore, the probability of line 39 being executed should be very low, since it is unlikely that two continuous feature values match exactly. Hence, we deemed it unnecessary to adjust the area of hyperrectangles for this case.

The original NGE paper also did not specify a policy for handling examples containing missing features. In the context of nearest neighbor algorithms, Aha [Aha90, Section 5.2.1] evaluated three methods for distance computation with missing features. We adopted his *Ignore* method. This is one of the simplest methods for dealing with missing features: If a feature of an example is missing, then the distance for that feature is 0. Furthermore, the total distance over all features is divided by the number of known features to distinguish a perfect match from a missing feature (both have distance 0).

We incorporated this methodology into the generalization procedure of NGE as follows: Whenever a hyperrectangle in NGE is extended to include an example with missing features, then the range of the hyperrectangle is extended to cover the entire input space for each missing feature (see lines 20, 25, and 27-32 in Figure 5.1).

5.2 Experiments on Parameter Sensitivity

We explored the sensitivity of NGE to its user-specified parameters. For NGE, the parameters of interest are (a) the number of starting seeds, (b) the treatment of un-

[11]The size of trivial point hyperrectangles is assumed to be 0.

generalized exemplars, and (c) the order of presentation of the examples. For kNN, the only parameter of interest is the number of nearest neighbors (k). See Chapter 4 for a discussion of kNN's sensitivity to the choice of k.

5.2.1 Number of starting seeds

Figure 5.2 shows the performance of NGE on the *quadrants*, *diagonal*, and *banded* tasks for several different numbers of starting seeds. The performance is shown relative to the performance of simple nearest neighbor. For the *quadrants* and the *diagonal* tasks, where the number of classes is small, NGE's performance is particularly poor for small numbers of seeds. This contradicts Salzberg's findings [Sal91] that the performance of NGE was not sensitive to the size of the seed set.

Not surprisingly, NGE performed better on the *quadrants* and *banded* tasks, where the decision-boundaries are axis-parallel, than on the *diagonal* task, where the boundary is diagonal (Figure 5.2). On the *diagonal* task, simple first-nearest neighbor outperformed NGE.

At the right end of the figure (over the label "cv"), we show the performance that is obtained if leave-one-out cross-validation [WK91] is employed to determine the optimal number of seeds. This strategy worked very well, so we adopted it in all subsequent experiments (unless otherwise noted).[12] The following number of seeds was tested during each leave-one-out cross-validation run: 3, 5, 7, 10, 15, 20, and 25. Cross-validation is inherently non-incremental, so a disadvantage of using cross-validation is that it destroys the incremental nature of NGE.

Note that if NGE is given a sufficiently large number of seeds, the algorithm becomes the simple nearest-neighbor algorithm. In the limit, there is one seed for every data point. This limit is not reached in these three tasks, however. NGE needed only approximately 6% (*quadrants* task), 13% (*diagonal* task), and 28% (*banded* task)

[12] Leave-one-out cross-validation is computationally very expensive for NGE since even the smartest implementation would have to process approximately $\frac{n(n-1)}{2}$ examples for each cross-validation run.

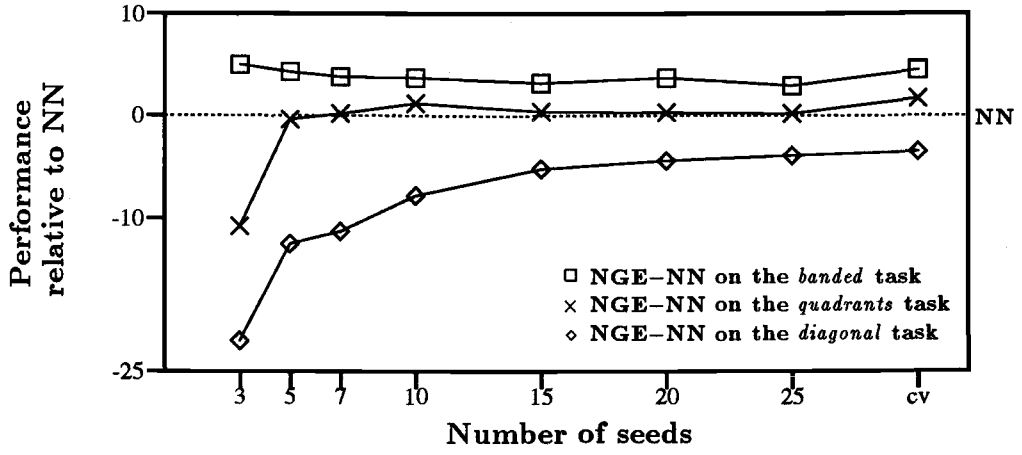


Figure 5.2. The performance of NGE relative to NN when NGE is initialized with varying numbers of seeds (cv: leave-one-out cross-validation). Base performance for NN is 97.6% correct in the *quadrants* task, 97.0% in the *diagonal* task, and 82.4% in the *banded* task. Data points represent means over 25 replications with 350 training examples and 150 test examples. See Table 5.6 for detailed numbers.

of the storage that was required by NN to store the entire training set (detailed numbers are provided in Table 5.6).

5.2.2 Treatment of ungeneralized exemplars

In NGE, hyperrectangles are initialized to points in the input space and should, therefore, have size 0 before they are generalized to non-trivial hyperrectangles. We have found that in the Led-7 Display domain, however, initialization of the size of hyperrectangles to 1 (the maximal size any hyperrectangle can have) led to a significant performance improvement (from 43.0±1.4% correct to 59.8±1.0%). This was an artifact of the Led-7 Display domain that specifically resulted from the fact that Led-7 Display has large numbers of training examples with identical feature vectors belonging to different classes. It that case, each query falls inside of several hyperrectangles and will be classified with the class of the smallest hyperrectangle it falls into. If the size of hyperrectangles is initially set to 0, then ungeneralized, trivial hyperrectangles cannot be distinguished from generalized, trivial hyperrectangles.

However, generalized, trivial hyperrectangles should be preferred over ungeneralized, trivial hyperrectangles, since they are more likely to be correct.

The initial size of hyperrectangles had no effect on NGE's performance in any of the other domains. In the experiments reported in the remainder of the paper, we chose to initialize the size of the hyperrectangles to 0, except in the Led-7 Display domain, where we initialized the size to 1.

5.2.3 Order of presentation of training data

NGE is sensitive to the order in which the training examples are presented. Table 5.1 shows the results of an experiment in which the training set/test set partitions were fixed while the order of presentation of the training set was randomly varied. We can see that the performance varies widely across these domains. This is a serious drawback of the NGE algorithm.

Table 5.1. The performance of NGE on one specific training/test set partition. Numbers shown indicate the performance of NGE when run on 25 random permutations of the same training set.

Domain	Mean	Median	Min	Max
Iris	91.8±0.8	93.3	84.4	97.8
Hungarian	78.0±0.5	77.3	71.6	83.0
Voting	93.3±0.4	93.1	89.2	96.2

Unfortunately, it is difficult to choose a "good" order for the training set. We could not find an effective way to apply cross-validation methods, for example, to select a good order. In the results reported below, a random order was selected for each run of NGE, and (as with all of the other algorithms) the mean of 25 runs is reported.

5.3 Comparison of NGE and kNN

The performance of three versions of the NGE algorithm is compared in Figure 5.3 to the performance of the k-nearest neighbor algorithm (kNN). The three NGE versions use different methods for choosing the number of initial seeds during training. In NGE_{cv} the number of seeds is chosen via leave-one-out cross-validation, $\text{NGE}_{3\ seeds}$ is always initialize with three seeds,[13] and in NGE_{limit} the number of seeds was increased to at most 50% of the training data. The rationale behind NGE_{limit} is that the amount of storage required for each hyperrectangle is twice the amount of storage required for a single data point. Hence, when the number of seeds equals 50% of the training data, the total space required by NGE equals the space required by kNN (assuming that similar methods for dealing with ties and missing features are used). Beyond that point, NGE has no data compression advantage over kNN.[14]

The k-nearest neighbor algorithm outperforms NGE_{cv} by a statistically significant amount in all of the eight non-constructed domains (Figure 5.3). In all domains, NGE_{cv} achieved a significant (i.e. between 60% and 85%) compression of the data. By substantially increasing the number of seeds (NGE_{limit} in Figure 5.3), it was possible to significantly improve the performance of NGE_{cv} in the *diagonal* task, and in the Led-7 Display, Cleveland, Hungarian, and Letter Recognition domains. However, NGE_{limit} is still significantly inferior to kNN in performance in all but one non-constructed domain. The drop in performance in the Voting domain and the *quadrants* task is due to the fact that in these domains, leave-one-out cross-validation over a small number of different seed set sizes is more beneficial than increasing the size of the seed set.[15] However, the improvement in performance by NGE_{limit} comes

---

[13]The results for the Iris domain differ slightly from those reported by Salzberg [Sal91], because accuracies obtained during leave-one-out cross-validation rather than repeated train/test partitions were reported. The results with leave-one-out are 95.3% for nearest neighbor and 92.8% for NGE.

[14]One could improve on this by allocating space for the lower and upper corner of each hyperrectangle only if the hyperrectangle is non-trivial.

[15]Leave-one-out cross-validation over more than, approximately, 10 different numbers

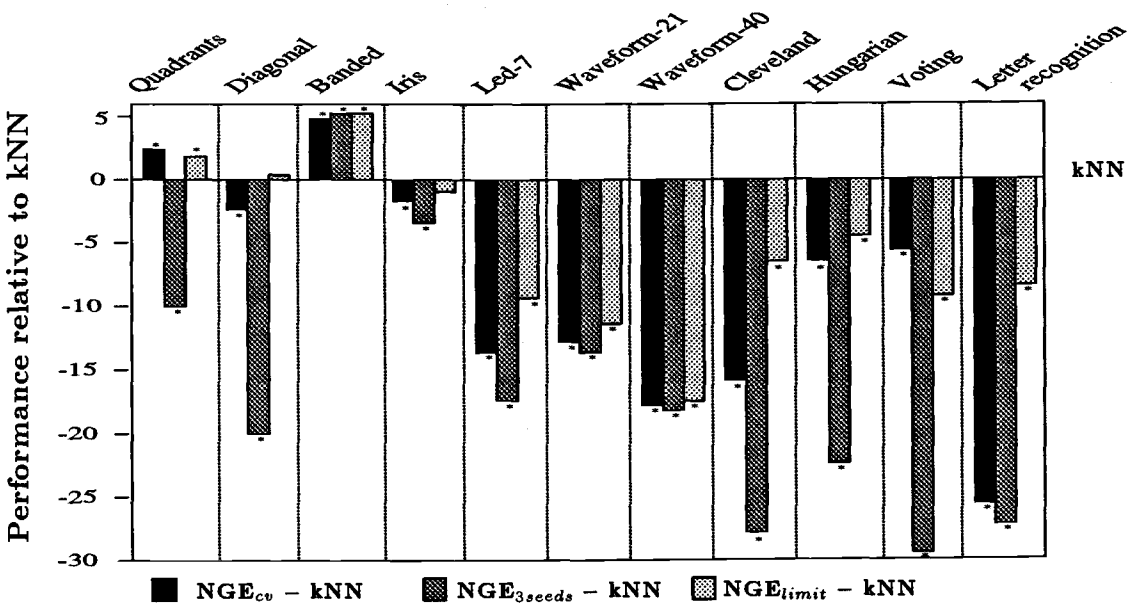


Figure 5.3. The performance of NGE$_{cv}$, NGE$_{3seeds}$, and NGE$_{limit}$ relative to kNN. Shown are percentage point differences between NGE$_{cv}$ and kNN, between NGE$_{3seeds}$ and kNN, and between NGE$_{limit}$ and kNN. These (*) differences are statistically significant. See Tables A.1 through A.5 for detailed numbers.

at a high cost: In all cases where NGE's performance improved, it also used more memory than kNN.

The learning curves for kNN and NGE have generally the shape that we expect from most inductive learning algorithms: Performance increases with the number of training examples and the increase levels off after the training set has reached a certain size (Figure 5.4). In the Waveform, Led-7, and Letter Recognition domains, the performance of NGE$_{cv}$ levels off much earlier than kNN's. Furthermore, the graphs for the Cleveland, Hungarian, and Voting domains show some erratic behavior for NGE$_{cv}$. In the Hungarian and Voting domains, NGE$_{cv}$ reaches its (near) peak performance after only 25 training examples have been seen. For more than 25 training examples, performance of NGE$_{cv}$ varies within two standard errors in these domains. In the Cleveland domain, the performance of NGE$_{cv}$ peaks also at 25 examples with 72.6±1.9% correct, but then drops down to 66.9±1.8%. Through

of seeds is not computationally feasible.

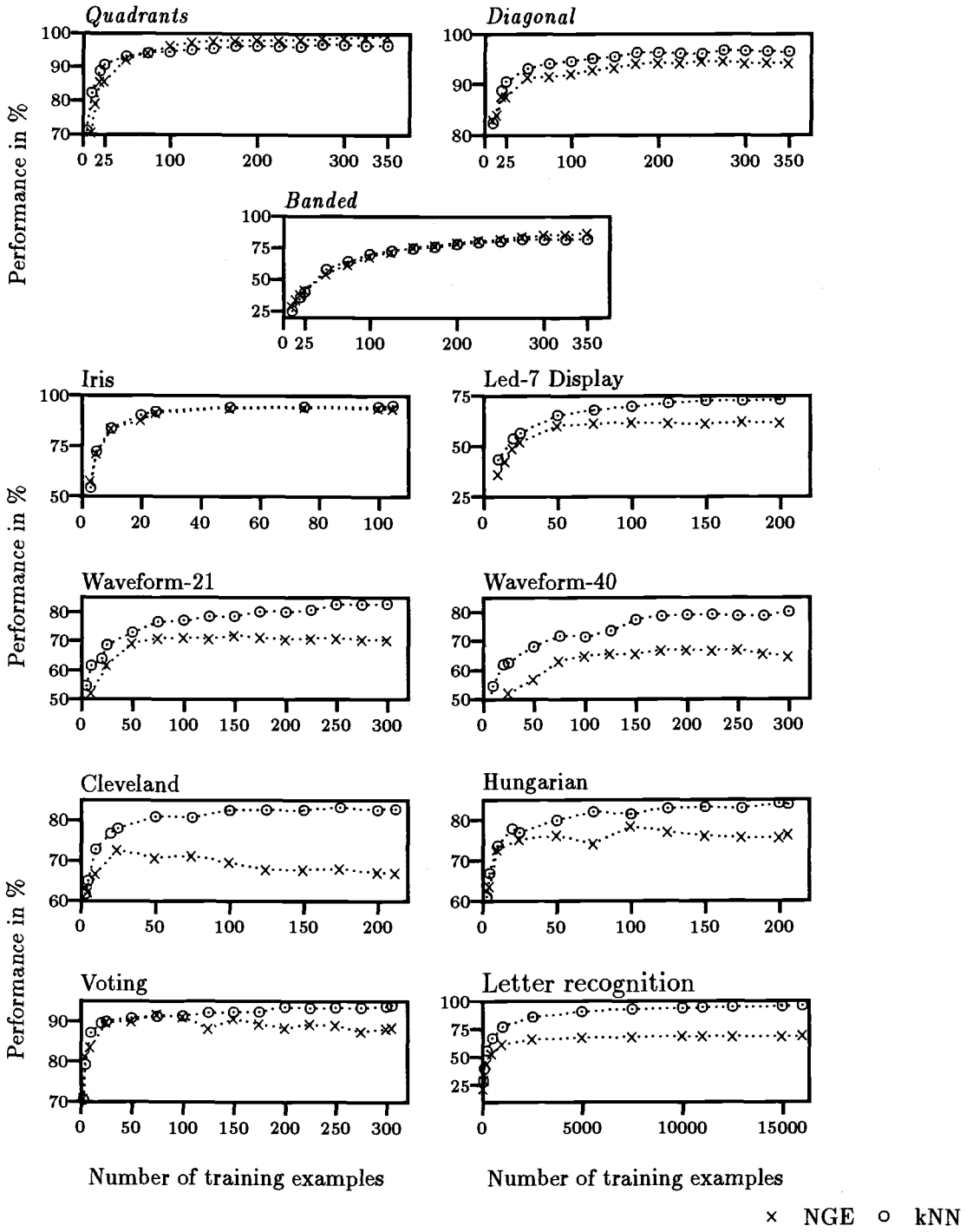


Figure 5.4. The performance of NGE and kNN for different numbers of training examples. Each data point denotes the mean of 25 experiments. Note the different scales on both axes of these graphs.

inspection of the number and sizes of hyperrectangles constructed by NGE_{cv} in these domains, we were able to determine the cause of this unusual behavior. The number of hyperrectangles stored by NGE_{cv} grows only sub-linearly with the number of training examples. Although that is a desirable property of any machine learning algorithm, it may cause problems for NGE_{cv}, since existing hyperrectangles may be generalized (extended) too often. This means that every time a hyperrectangle is enlarged it may actually become less relevant. We conclude that this behavior constitutes a serious deficiency in NGE's search and generalization procedure.

5.4 Possible Explanations for Inferior Performance of NGE

Given the close relationship between NGE and kNN, it is surprising that NGE performs so much worse than kNN. With any learning algorithm, there can be two fundamental sources of problems. First, the bias of the algorithm may be inappropriate for the application domains. Second, the implementation of that bias may be poor (e.g., because poor search algorithms are employed).

Salzberg never formally defines the bias of NGE. Let us define it to be "find the minimum number of axis-parallel hyperrectangles (possibly nested or overlapping) that correctly classifies the training data."

There is some evidence that this bias is inappropriate. We know that in the *diagonal* task (non-axis-parallel decision boundary), the axis-parallel bias is inappropriate (see Figure 5.2), but this is an artificially-constructed domain. However, Aha [Aha90] reports the performance of C4.5 [Qui92] in six of the domains which are also used in this thesis. C4.5 also has a rectangular bias and performs, under similar conditions, significantly better than NGE in these six domains (Aha [Aha90], Section 4.3.3).[16] This suggests that the axis-parallel bias is not the cause of NGE's poor performance.

[16]There are many other differences between NGE and C4.5. However, Aha's results indicate that a rectangular bias may be of no hindrance given the proper search algorithm.

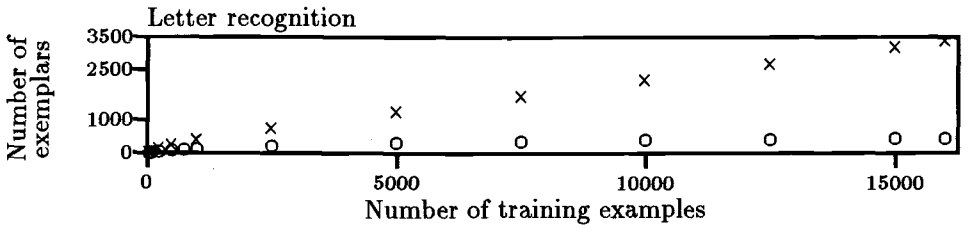


Figure 5.5. Number of exemplars stored by NGE when trained with 25 seeds on differently sized training sets from the Letter recognition task. Shown is the total number of hyperrectangles that were stored during training (×) and the number of hyperrectangles which were generalized at least once (o). Each data point denotes the mean over 25 experiments.

Examination of the learned hyperrectangles in several of the other domains suggests that permitting rectangles to nest and overlap is a problem. The most common form of nesting is that a large, generalized hyperrectangle is created and then many single-point rectangles are nested inside it as exceptions. This can be seen in Figure 5.5, which plots the number of hyperrectangles created and the number that are actually generalized to be non-point rectangles. We can see that the overwhelming majority of hyperrectangles are never generalized.

These single-point hyperrectangles are virtually never used for classifying new test examples, because if a test example falls inside a large hyperrectangle, the distance to that hyperrectangle is zero. A single-point hyperrectangle will not be used unless either (a) the test example exactly coincides with the single-point rectangle or (b) the single-point rectangle is not nested inside another rectangle.

NGE also permits generalized rectangles to overlap, even if they don't nest. This may be a problem as well. One situation in which overlapping rectangles will be created is if the distributions of examples from two classes, A and B, overlap. The optimal decision rule (under a uniform loss function [DH73]) is to place the decision boundary at the point where the probability density of examples from class A equals the probability density of examples from class B. However, NGE instead arbitrarily assigns all examples in this overlapping region to one of the classes—the one which has the smaller rectangle.

In addition to these hypotheses about the bias of NGE, there is considerable evidence that the bias is not implemented well by NGE's incremental heuristic procedure. From the sensitivity experiments, we know that NGE is very sensitive to the order in which the training examples are presented. For some orders, it does very well.

In the *quadrants* and the *banded* task, we can determine the optimal set of hyperrectangles by inspection (4 and 10, respectively). NGE does not find this optimal solution, but instead constructs an average of 10.8±1.1 and 49.6±1.2 rectangles (Figure 5.6). In the *diagonal* task, on the other hand, the optimal solution involves a large number of rather small, overlapping rectangles (one for every training example that lies near the decision boundary). However, NGE does not find this solution either. It constructs some rectangles that are too large, and then nests the smaller ones in them (Figure 5.6).

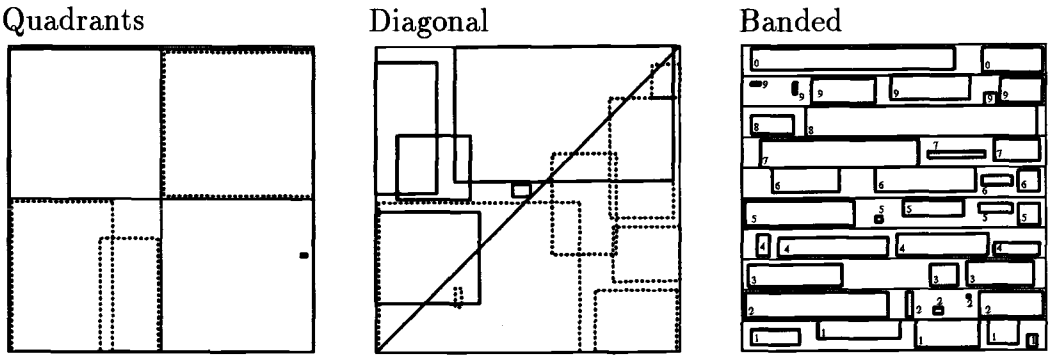


Figure 5.6. Rectangles constructed by NGE_{cv} in the *quadrants*, *diagonal*, and *banded* tasks in one representative experiment. In the *quadrants* and *diagonal* tasks, dashed (solid) lines indicate the location of rectangles representing positive (negative) examples. In the *banded* task, digits indicate the class each rectangle represents. Trivial (point) rectangles not displayed. Note that in the *quadrants* task a single rectangle of class 0 covers the entire input space.

In summary, we have three hypotheses that can explain why NGE is performing poorly relative to kNN:

H1. nested rectangles,

H2. overlapping rectangles, and

H3. poor search algorithm.

To test these hypotheses, we conducted a series of experiments in which we modified NGE to eliminate one or more of these suspected problems and measured the resulting change in performance.

In the first experiment, we tested H1 by modifying NGE so that it produced relatively few nested rectangles but still permited overlapping rectangles. We did not otherwise change the search procedure.

In the second experiment, we tested H2 by modifying NGE so that it produced no overlapping rectangles of different classes (with the exception of rectangles entirely nested inside one another). We did not otherwise change the search procedure.

In the third experiment, we tested H3 by making a simple modification to incremental NGE to improve upon the second-match heuristic, with the goal of finding fewer hyperrectangles.

Finally, in the fourth experiment, we tested all of the hypotheses simultaneously by implementing an entirely different search procedure that completely eliminated nested rectangles and overlapping rectangles and also reduced the total number of rectangles constructed.

These experiments and their results are described in the remainder of this section.

5.4.1 Greedy NGE (avoid nesting)

To test H1, it is necessary to construct a variant of NGE that avoids nesting rectangles. A major cause of nested rectangles is the second match heuristic (line 9 in Figure 5.1). If the nearest rectangle is of the wrong class but the second nearest

rectangle is of the right class, then the second-nearest rectangle is expanded to cover the new example. In many cases, it will also cover the nearest rectangle (which could be a single point), and thus create nesting.

Salzberg [Sal91, Section 3.5] introduced and tested a version of NGE, called Greedy NGE, that does not have the second match heuristic. This greedy version stores an example as a new hyperrectangle whenever the closest previously stored hyperrectangle is of a different class than the example. According to Salzberg, the second match heuristic in NGE is necessary to construct nested or overlapping hyperrectangles. This is not true: NGE may still construct overlapping or nested hyperrectangles even if its second match heuristic is disabled, because it can "grow" a hyperrectangle until it overlaps or covers another hyperrectangle. In fact, Greedy NGE did construct overlapping hyperrectangles (quite frequently) and nested hyperrectangles (in a few cases) in the experiments that we conducted (data not shown).

The predictive accuracy of Greedy NGE was significantly better than NGE's in three domains (Cleveland, Hungarian, and Voting) and significantly worse in 4 others (the *quadrants* and *banded* tasks, and the Waveform-21, and Waveform-40 domains). The results in the *banded* task and the Waveform-40 domain were particularly poor (Figure 5.7).

Based on these, there is not much evidence that nested rectangles are a major problem for NGE.

5.4.2 NGE without Overlapping Hyperrectangles (NONGE)

To test H2, we want to construct a variant of NGE that avoids overlapping rectangles. This can be accomplished as follows. Let us define P to be the potential new hyperrectangle that is constructed by the calls to "generalize" in lines 9 and 10 of Figure 5.1. Rectangle P is the rectangle formed by extending either the first match or the second match rectangle so that it covers the training example.

In No-Overlap NGE (NONGE), we construct P and then check whether it would intersect with any hyperrectangle from any other class. If P would intersect another

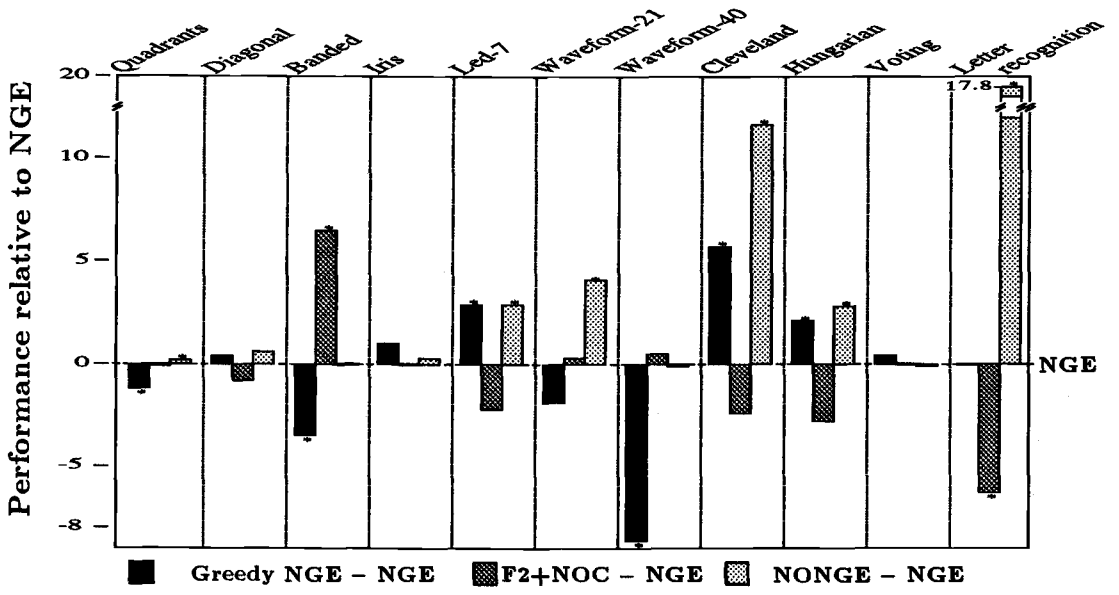


Figure 5.7. Performance of Greedy NGE, NGE with an additional matching heuristic (F2+NOC: first two matches and the nearest exemplar from the examples own class are considered), and NONGE relative to NGE. These (*) performance differences between NGE and its modifications are statistically significant ($p < 0.05$). See Tables A.1 through A.5 for detailed numbers.

rectangle, then we reject P and create a new, single-point rectangle instead. However, if P would be completely contained within another hyperrectangle, we accept P. This way, nested rectangles are permitted, but overlapping (non-nesting) rectangles are forbidden.

In Figure 5.7, we see that NONGE was significantly better than NGE in 6 of the 11 domains, and it was never significantly worse than NGE. This strongly supports hypothesis H2—that overlapping rectangles cause problems for NGE.

5.4.3 A better merge heuristic for NGE?

NGE stores a training example as a new hyperrectangle whenever the two nearest hyperrectangles have different output classes than the example. In some cases, however, this can create unnecessary new rectangles. Consider Figure 5.8. Here, rectangle C is further away from point P than either rectangle A or rectangle B. However, because

rectangle C has the same class as point P, it could be extended to cover point P without overlapping either of rectangles A or B. By extending rectangle C in this way, we avoid creating a new generalized exemplar for point P.

We developed a modified version of NGE, called F2+NOC, that detects this situation. If the first two matches (to the nearest and second-nearest hyperrectangles) fail, F2+NOC finds the nearest hyperrectangle having the same class as the example. It then extends that nearest hyperrectangle to include the new example if the expanded hyperrectangle would not cover any hyperrectangles from any other classes. Otherwise it stores the example as a new hyperrectangle. This gives NGE another chance to generalize and should, in general, reduce the amount of memory required by NGE.

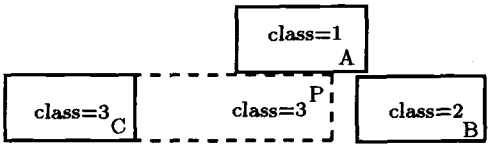


Figure 5.8. Example showing that rectangle C can be extended to cover point P

F2+NOC can be considered as a weak test of hypothesis H3 (that the search algorithm of NGE needs improvement). Tables A.1 through A.5 indicate that this additional matching heuristic indeed achieves a reduction in storage in most domains. Hence, it is a better implementation of the NGE bias. However, as shown in Figure 5.7, F2+NOC performs significantly better than NGE only in the *banded* task, while the reduction in storage is directly related to a loss in predictive accuracy in five domains.

Hence, this improvement to NGE's search algorithm does not explain the poor performance of NGE relative to kNN.

5.4.4 Batch NGE

To obtain a better test of H3, we constructed two batch algorithms (OBNGE and BNGE) for the NGE bias. These algorithms begin with all training examples in memory as point hyperrectangles and progressively merge them to form generalized hyperrectangles. At each step, the two hyperrectangles are merged subject to one of the following constraints:

OBNGE Only merge if that merge would not cause misclassification of any training examples. This algorithm requires testing of the entire training set for each potential merge. It permits overlapping but no nesting of rectangles. We call it OBNGE (Overlapping Batch NGE).

BNGE Only merge if the new hyperrectangle does not cover (or overlap with) any hyperrectangles from any other classes. This algorithm requires intersection of each potential merge with all hyperrectangles from all other classes. It does not permit overlapping or nesting. We call it BNGE (Batch NGE).

The merging process in both algorithms is repeated until no more merges can be found (Figure 5.9). Note that these algorithms are somewhat dependent on the order in which potential merges are considered. They are greedy in that a merge is accepted as soon as the above mentioned conditions are satisfied. The *break* statement on line 60 of Figure 5.9 ensures that the rectangles representing different classes are grown at approximately equal rates. It is necessary to balance the growth of the rectangles to avoid generating hyperrectangles from a single class that would cover most of the input space. These overly large hyperrectangles would then prevent hyperrectangle from the other classes from growing. For reasons of computational complexity, arbitrarily chosen hyperrectangles from each class are grown alternatively until they cannot be extended further. An alternative search strategy for BNGE will be discussed in Section 5.5.

These algorithms are more conservative in generalizing beyond the training data than the original NGE algorithm, since they generate hyperrectangles only in those

```
43. Initialization:
44.   for all training examples E:
45.       q[class_of(E)].push(E);
46. Training:
47.   repeat
48.     for each output class C                          /* skip class C if q[C] is empty */
49.       T = q[C].pop();
50.       i = 1;
51.       if (q[C].empty()) final[C].push(T);
52.       while (not(q[C].empty()))
53.         N = q[C].ithClosest(T,i);          /* N is the hyperrectangle in q[C] that has the */
54.                                                        /* ith smallest distance to T. */
55.                              /* Check if the hyperrectangle built by T and N would overlap */
56.                                        /* with any hyperrectangles from any other classes. */
57.         if (not(overlaps(hyperrectangle(T,N),otherClasses(C))))
58.           q[C].remove(N);
59.           q[C].push(hyperrectangle(T,N));
60.           break;                                                  /* move on to next class */
61.         else
62.           i = i + 1;
63.           if (i > q[C].length())                  /* There exists no other hyperrectangle */
64.                                                          /* That T could be merged with */
65.             final[C].push(T);                                    /* Add T to final queue */
66.             T = q[C].pop();              /* Try to merge next hyperrectangle in this class */
67.             i = 1;
68.   until (q[C].empty() for all C)
```

Figure 5.9. Pseudo-code describing the construction of an BNGE classifier.

parts of the input space which clearly belong to a certain class. Furthermore, due to the fact that BNGE and OBNGE repeatedly pass over the training data, they may also significantly reduce the number of hyperrectangles that remain at the end. BNGE is also faster and easier to use than NGE, since no cross-validation of free parameters is required. OBNGE requires an organization of the training data in k-d trees [FJF77], for example, such that the cost of repeatedly testing the entire training set is minimized.

Numbers displayed in Figure 5.10 and Tables A.1 through A.5 show that BNGE significantly outperformed NGE in 7 of the 11 domains tested. In all cases, the performance of BNGE was better than NGE. On the other hand, OBNGE was significantly better than NGE only in the *banded* task, and significantly worse than NGE in three domains (the *quadrants* and *diagonal* tasks, and the Hungarian domain).

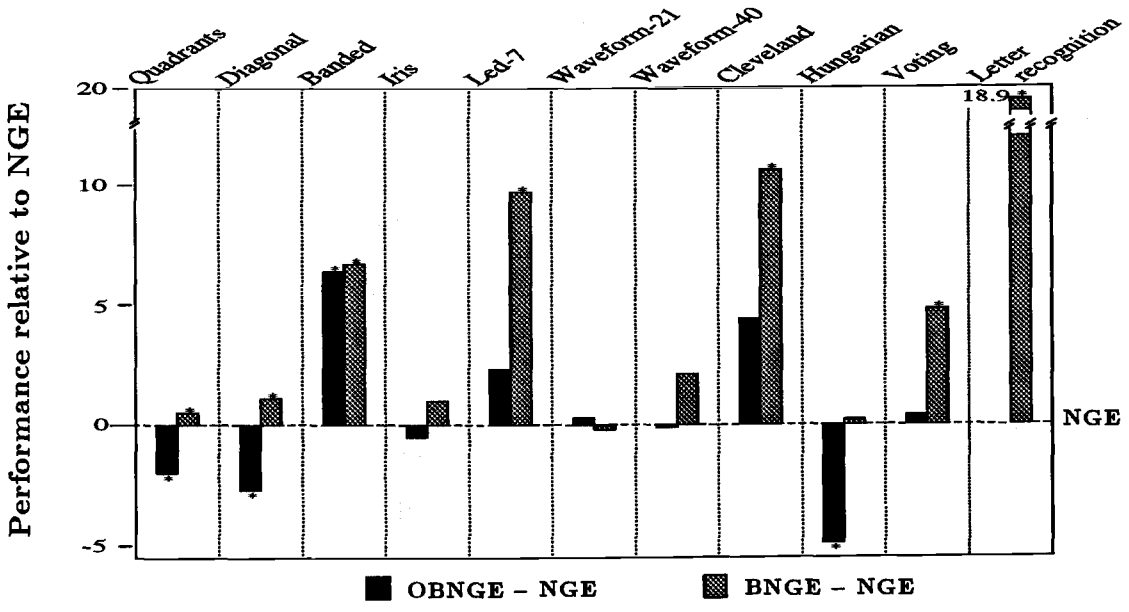


Figure 5.10. Performance of OBNGE and BNGE relative to NGE. Shown are percentage point differences between OBNGE and NGE and between BNGE and NGE. A * indicates that the performance difference between NGE and its modification is statistically significant ($p < 0.05$). See Tables A.1 through A.5 for detailed numbers.

This provides additional strong evidence that overlapping rectangles are an inappropriate bias for these domains (H2).

To test H3, we can examine first whether BNGE implements a better search algorithm. For the *quadrants* task, BNGE attains the optimal solution of 4 hyperrectangles. Furthermore, BNGE also uses only one hyperrectangle to cover the Iris Setosa class in the Iris domain. This is good evidence for the quality of the BNGE search procedure.

The incremental version of NGE most similar to BNGE is NONGE (NGE without overlapping rectangles). By comparing Figures 5.7, 5.10, and Tables A.1 through A.5, it can be seen that BNGE out-performed NONGE in four domains, while NONGE out-performed BNGE in two domains. This gives only weak evidence that the improved search algorithm of BNGE is responsible for the improved performance.

5.4.5 Discussion

From these experiments, we can see that there is weak support for H3 and strong support for H2 as explanations for the poor performance of NGE relative to the nearest neighbor algorithm. There is no support for H1. Versions of NGE that do not permit overlapping rectangles perform consistently better than NGE in all domains tested. The batch algorithm, BNGE, which does not permit nested or overlapping rectangles of different classes, performs quite well and avoids the need to choose the number of seeds for NGE by cross-validation.

5.5 Batch NGE revisited

The search algorithm employed by BNGE during training (Figure 5.9) is a very ad-hoc procedure, since it randomly picks one training example from each class and grows these (while alternating between classes) to the largest possible sizes. In some domains the algorithm will, therefore, construct a small number of very large rectangles in each class and will be left with some number of very small rectangles. There are many alternative methods of constructing a set of non-overlapping rectangles from a set of training examples. Finding the optimal set of rectangles with respect to some optimality criterion such as smallest combined size or smallest total number of rectangles is NP-complete. One method that may construct a better set of rectangles is Omohundro's best-first model merging technique [Omo92]. This technique dynamically chooses the structure of a classifier by iteratively combining "small" subcomponents of the classifier into larger components that typically cover a larger part of the input space and therefore more training examples. The process of merging is repeated until some criterion is satisfied. The best-first model merging technique can be used to construct a BNGE classifier (Figure 5.11). We call this algorithm BNGE_{MM} for BNGE with model merging. The basic idea behind the algorithm is to initially construct all hyperrectangles that would result from merging each training example with every other training example of its class and to sort these candidate hyperrectangles into a priority queue according to their sizes. Each element of this

queue is then either accepted into the final classifier or rejected. Two means for accepting potential hyperrectangles into the final classifier are described. The first method accepts hyperrectangles only if they have no contact whatsoever with any hyperrectangles from other classes (nesting_allowed = FALSE). The second methods allows nested hyperrectangles under certain circumstances. This version may increase BNGE's performance in the presence of noise.

5.5.1 Pruning

One of the main advantages of NGE and its variations when compared to the Nearest Neighbor algorithm is that NGE often finds a more compact representation of the data. For example, if all training examples of one class can be described by a single rectangle, then BNGE will find that rectangle. Often, however, NGE and BNGE store trivial point-hyperrectangles. Since these hyperrectangles cover no significant part of the input space, they may contribute little to the generalization accuracy of NGE while using up memory and slowing down the classifier during classification.

Pruning of exemplars that cover only one training example (i.e. were never generalized, denoted by BNGE_{p1}) had no significant effect on the performance of BNGE in any of the eleven domains used in the experiment described in Figure 5.12. A significant reduction in storage was observed in all domains except in the *quadrants* task, where BNGE had already found the smallest possible representation of the training data without pruning. Across all other domains, approximately 30% of the hyperrectangles stored by BNGE covered only a single training example and were removed by BNGE_{p1} (Table 5.2).

Pruning of exemplars that cover only two training examples (denoted by BNGE_{p2}) leads to a statistically significant decrease in performance in the *diagonal* task and the Letter Recognition domain (letters B and R, only). A further reduction in the storage required was achieved by BNGE_{p2} when compared to BNGE_{p1} (again, with the exception of the *quadrants* task). The amount of compression of the original

```
69.  Initialization:
70.   for all training examples E:
71.    for all other training examples E_N:
72.      H = hyperrectangle(E,E_N)
73.      priority_queue.insert(H)                    /* smallest elements first */
74.    final_queue.insert(E);

75.  Training:
76.   while (not(priority_queue.empty()))
77.    T = priority_queue.pop();
78.    if (merge_can_be_accepted(T, final_queue))
79.      final_queue.insert(T);
80.      final_queue.remove(covered_by(T) and same_class_as(T));
81.      for all elements E in priority_queue:
82.        if (class(E) == class(T)
83.          H = hyperrectangle(T,E)
84.          priority_queue.insert(H)

85.  merge_can_be_accepted(T, final_queue){
86.    if (T does overlap with any hyperrectangles from other classes)
87.      return FALSE;
88.    else if (T does not cover any hyperrectangles from other classes)
89.      return TRUE;
90.    else if (nesting_allowed){
91.        Let T_1 and T_2 be the hyperrectangles T is composed of
92.        s = number_of_training_examples_covered_by(T_1)
93.        t = number_of_training_examples_covered_by(T_2)
94.        m = min(s,t)
95.        S = 0;
96.        for all hyperrectangles H covered by T in final_queue with class(H) ≠ class(T)
97.          S += sum(number_of_training_examples_covered_by(H)
98.        if (S < m)
99.          return TRUE;
100.   return FALSE;
101. }
102.
```

Figure 5.11. Pseudo-code describing the construction of an BNGE classifier with best-first model-merging.

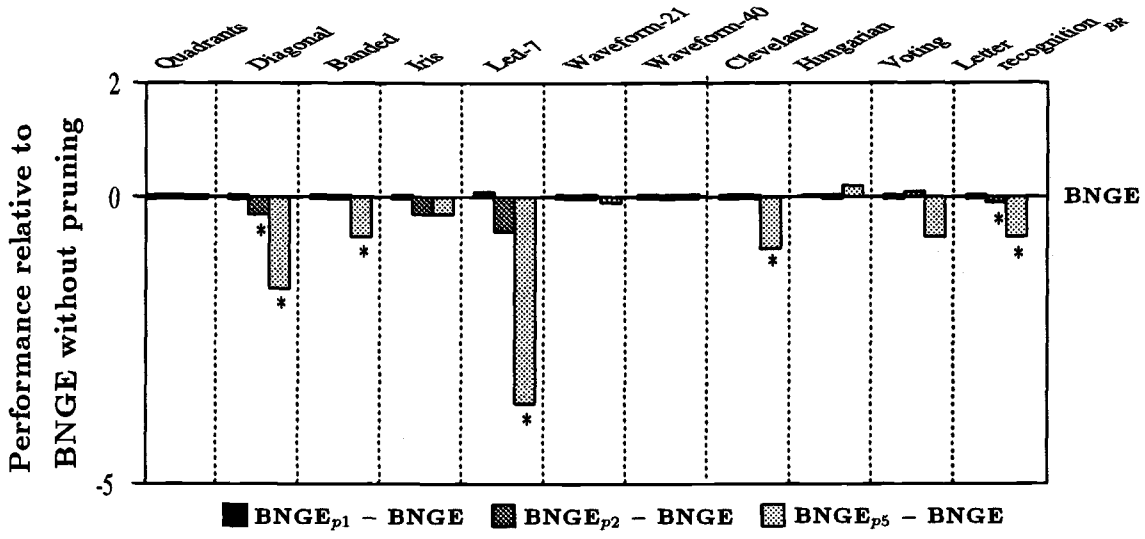


Figure 5.12. Performance differences between BNGE without pruning and BNGE with different levels of pruning on the test set. The subscript px indicates that hyperrectangles which cover at most x training examples were removed before the classifier was tested. Performance relative to BNGE without pruning is shown. These differences (*) are statistically significant ($p < 0.05$).

training data achieved by BNGE$_{p2}$ was substantial. Its data compression rate was more than 80%[17] for all domains in which it was tested.

Larger levels of pruning can lead to an unacceptable degradation in performance in some domains and is not be recommended unless memory (and computational resources) are very sparse.

It is important to note that the main purpose of the pruning technique described here is to find a more compact BNGE classifier with somewhat similar classification accuracy. Since this approach is a modification of BNGE's bias, it may also suffer from the same problems as other pruning techniques [Sch93a] with respect to classification accuracy. However, pruning never increases the amount of storage required by BNGE.

Pruning of hyperrectangles that cover only a few training examples can also be used to obtain a simplified description of the training data. This description may then

[17]Note that each hyperrectangle consumes approximately twice as much memory as a single training example.

Table 5.2. Number of hyperrectangles stored by BNGE without pruning (column 1) and when different levels of pruning were employed. Numbers in parentheses indicate ratio of the number of hyperrectangles stored to the number of training examples. The subscript px indicates that hyperrectangles which cover no more than x training examples were removed before the classifier was tested.

Domain	BNGE	BNGE_{p1}	BNGE_{p2}	BNGE_{p5}
Quadrants	4.0±0.0 (1%)	4.0±0.0 (1%)	4.0±0.0 (1%)	4.0±0.0 (1%)
Diagonal	19.7±0.5 (6%)	15.0±0.3 (4%)	13.2±0.3 (4%)	10.1±0.3 (3%)
Banded	24.0±0.6 (7%)	14.9±0.3 (4%)	13.0±0.3 (4%)	11.3±0.2 (3%)
Iris	7.0±0.3 (7%)	4.8±0.2 (5%)	3.8±0.1 (4%)	3.2±0.1 (4%)
Cleveland	38.6±0.5 (18%)	25.5±0.6 (12%)	18.5±0.4 (9%)	10.0±0.4 (5%)
Hungarian	39.0±0.6 (19%)	27.0±0.5 (13%)	19.0±0.5 (9%)	9.2±0.2 (4%)
Voting	22.7±0.8 (7%)	14.8±0.3 (5%)	10.8±0.3 (4%)	6.5±0.2 (2%)
Waveform-21	44.3±0.8 (15%)	29.6±0.9 (10%)	21.0±0.7 (7%)	10.3±0.4 (3%)
Waveform-40	47.0±0.9 (16%)	30.7±0.8 (10%)	20.2±0.6 (7%)	9.2±0.4 (3%)
Led-7 Display	72.2±1.0 (36%)	31.8±0.6 (16%)	17.6±0.4 (9%)	9.4±0.2 (5%)
Letter Recog.$_{BR}$	62.2±1.4 (6%)	46.4±0.9 (5%)	37.3±0.8 (4%)	24.1±0.6 (2%)

be used to evaluate the amount of information that each input feature carries. For example, we inspected the hyperrectangles in the Hungarian domain that remained after all hyperrectangles covering five or fewer training examples were removed from the classifier (denoted by BNGE_{p5} in Figure 5.12). From these hyperectangles we could determine that 4 of the 13 input features in this domain were completely irrelevant. These hyperrectangles could be further interpreted to describe the typical patient who is likely to suffer from heart disease as a middle-aged male experiencing atypical angina or asymptomatic chest pains with exercise-induced angina and a medium to high ST depression induced by exercise relative to rest. After pruning in the Voting Records domain., only one hyperrectangle for Republicans and one for Democrats was left to describe the voting patterns of the members of the US congress in the legislative period described in that data set. In particular, the votes

on adoption of the budget and the physician fee freeze were most informative, and 11 of the 16 features were irrelevant.

5.6 A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm

The evaluation of the hypotheses regarding the reasons for the relatively poor performance of NGE when compared to kNN has led to the design of an improved NGE algorithm that we termed BNGE. This algorithm outperformed NGE in 7 of the 11 domains tested (Figure 5.10); It also outperformed the first-nearest neighbor algorithm in 5 of the 11 domains, and was outperformed by NN in only two domains (Figure 5.13). However, when compared to the k-nearest neighbor algorithm, BNGE's performance is still significantly inferior to kNN's in 7 of the 11 domains and superior to kNN's in only 2 domains (Figure 5.13)

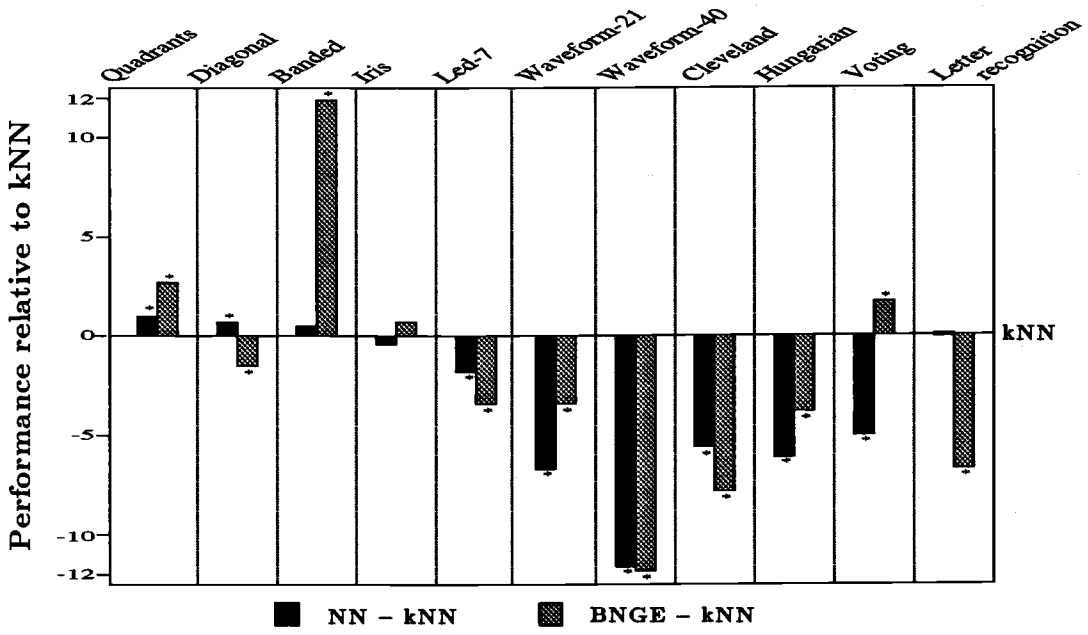


Figure 5.13. Performance of NN and BNGE relative to kNN. An * indicates that the performance difference between kNN and the other algorithms is statistically significant ($p < 0.05$). See Tables A.1 through A.5 for detailed numbers.

The following experiment was conducted to determine the cause of this remaining performance difference: the accuracy of the classifications assigned to the test cases

Table 5.3. Comparison of correctness of classifications made by BNGE inside versus outside of hyperrectangles. Numbers are based on a single repetition.

	Percentage of test examples			
	classified		classified	
Domain	inside	of these incorrect	outside	of these incorrect
Quadrants	93.3	0.0	6.7	20.0
Diagonal	80.7	4.1	19.3	10.3
Banded	76.0	0.9	24.0	38.9
Iris	80.0	0.0	20.0	22.2
Led-7	89.4	26.0	10.6	56.6
Waveform-21	26.0	11.5	74.0	33.8
Waveform-40	12.0	8.3	88.0	35.2
Cleveland	30.8	3.6	69.2	33.3
Hungarian	45.5	12.5	54.5	22.9
Voting	83.0	2.8	17.0	13.6
Letter recognition	68.4	1.0	31.6	32.8

by BNGE was measured separately for queries that were covered by at least one hyperrectangle and those that fell outside of all hyperrectangles. Results indicate that BNGE's predictions are substantially more often correct for queries that are inside of hyperrectangles than for queries that are outside of all hyperrectangles (Table 5.3). Displayed in Table 5.3 are the percentages of test examples that were covered by at least one hyperrectangle (column 2), the percentage of these test examples that were misclassified (column 3), the percentage of test examples that were outside of all hyperrectangles (column 4), and the percentage of these "outside"-test examples that were misclassified (column 5). The table clearly shows that BNGE commited more errors when predicting the class of test examples that were not inside any hyperrectangles, than when predicting the class of test examples that were inside hyperrectangles. Thus, we decided to use a different classifier for any queries that were not covered by hyperrectangles.

Table 5.4. Values of k used by KBNGE. Leave-one-out cross-validation was used in each of the 25 repetitions of the experiments to determine the value of k.

	k value		
Domain	min	max	average
Quadrants	1	99	24.7±6.7
Diagonal	1	27	6.5±1.5
Banded	1	5	1.6±0.2
Iris	1	18	8.0±0.8
Led-7 Display	2	7	4.3±0.4
Waveform-21	7	92	34.4±4.2
Waveform-40	14	93	43.3±5.0
Cleveland	3	57	18.6±3.5
Hungarian	21	57	37.2±2.3
Voting	3	10	6.1±0.5
Letter recognition	1	1	1.0±0.0

In the experiments described in Figure 5.15, the k-Nearest Neighbor algorithm was used as that classifier.[18] Un-generalized exemplars were pruned to accelerate the classifier. We call this hybrid method KBNGE to indicate that it is a combination of BNGE_{p1} (as defined in Section 5.5.1) and kNN.

The KBNGE algorithm has two main advantages over kNN: (1) Areas that clearly belong to only one class are represented by only one hyperrectangle. This can often lead to significantly faster classification times. The computationally more expensive kNN classifier is only used to classify queries in areas with complex decision boundaries or high levels of noise. (2) The hyperrectangles constructed can be inspected and interpreted by the user. This quality of user interpretability may lead

[18]Once again, k values were determined via leave-one-out cross-validation [WK91]. Values of k varied significantly across domains and for different random partitions of the training data within most domains (see also Table 5.4).

to a greater acceptance of decisions made by KBNGE as compared with those made by kNN or neural networks.

KBNGE is faster than kNN at classification time if the following rough formula is satisfied:

$$\#(\textit{training examples}) \quad > \quad 2 \times \#(\textit{hyperrectangles}) + (1 - x) \times \#(\textit{training examples}) \qquad (1)$$

where x is the percentage of test cases classified by BNGE. All other $(1 - x)\%$ of the test cases are classified by the kNN classifier. The value of x differs from domain to domain (see Table 5.3) and must be determined empirically. The justification for the formula is that kNN has to compare each query to all training examples, while KBNGE must compare each query to all hyperrectangles (a comparison to a hyperrectangle is approximately twice as expensive as a comparison to a training example), and if the query is not covered by any hyperrectangles (which happens in $(1 - x)\%$ of the cases), KBNGE must compare the query to all training examples.[19] Formula (1), evaluated with the data displayed in Table 5.3, shows that KBNGE is faster than kNN in all domains tested. In domains with large amounts of noise (Waveform, Cleveland, and Hungarian) kNN is used very often (Figure 5.14), which indicates that a noise tolerant version of BNGE should help to improve the speed of KBNGE even further.

The number of training (and test) examples covered by any hyperrectangle differs widely within and across domains. For example, a single hyperrectangle is always constructed in the Iris domain to cover all instances of Iris Setosa, while in the Cleveland and Hungarian domains a single hyperrectangle never covers more than approximately 30% of the training (20% of the test) examples of its class.

The classification accuracies achieved by KBNGE are comparable to those of kNN in most domains, and superior or indistinguishable from the accuracies obtained

[19]Formula (1) assumes that retrieval of training data is not conducted more efficiently with methods such as k-d trees [FJF77] or box-trees [Omo89]. In domains with many relevant features, neither k-d trees nor box-trees provide significant speedups over serial search. In domains where they do provide speedups, KBNGE could also be accelerated by storing the hyperrectangles in a box tree.

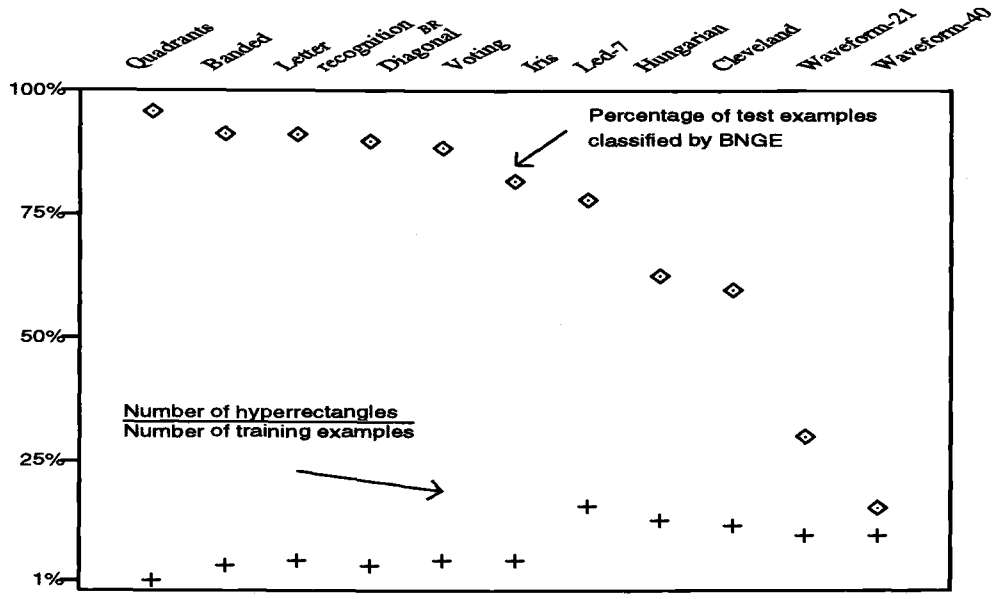


Figure 5.14. Ratio of test examples classified by the $BNGE_{p1}$ part of the KBNGE classifier ($\diamond$). All other test examples were classified by the kNN part of KBNGE. The ratio of hyperrectangles stored by $BNGE_{p1}$ to the number of training examples (+) is shown for comparison.

by BNGE in all domains (Figure 5.15). The rectangular bias of BNGE enables KBNGE to significantly outperform kNN in domains where that bias is most appropriate (the *quadrants* and *banded* tasks). A superior accuracy is also achieved in the Voting domain. The inferior performance of KBNGE (when compared to kNN) in the *diagonal* task, the Cleveland and the Hungarian domains, and in the Letter Recognition domain (letters B and R, only) indicates that BNGE's strong bias may lead to over-generalization in some domains. In the *diagonal* task, for example, BNGE generates rectangles with empty corners that reach into the space that belongs to the other class (Figure 5.16). Future should therefore investigate how this over-generalization can be avoided.

To summarize, the hybrid algorithm of kNN and BNGE successfully combines the classification accuracy of kNN and the classification speed of BNGE.

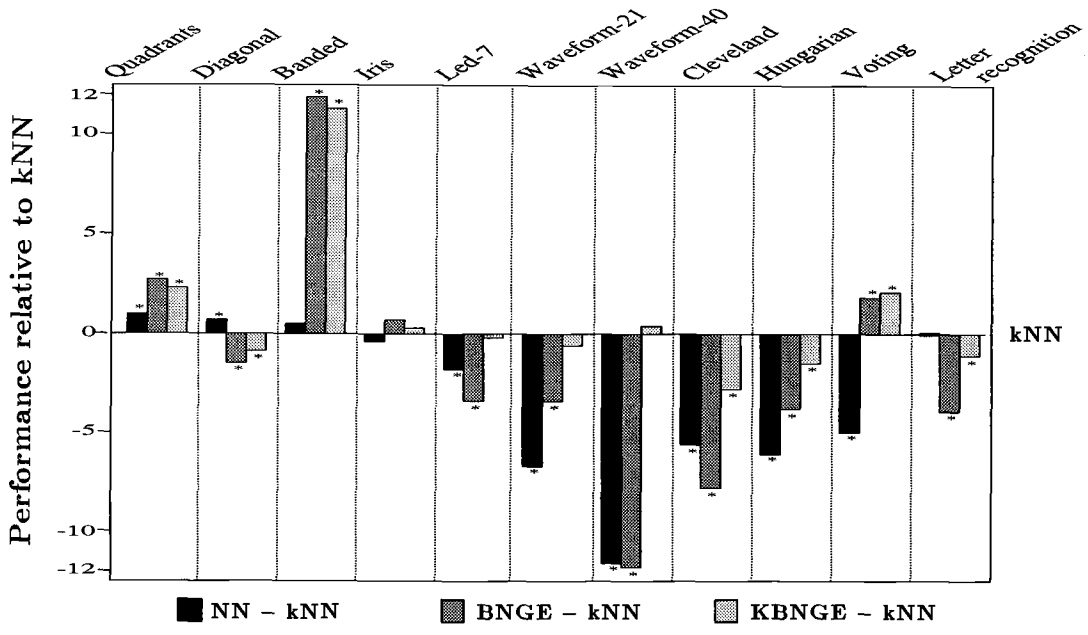


Figure 5.15. Performance of NN, BNGE, and KBNGE relative to kNN. An $*$ indicates that the performance difference between kNN and the other algorithms is statistically significant ($p < 0.05$). See Tables A.1 through A.5 for detailed numbers.

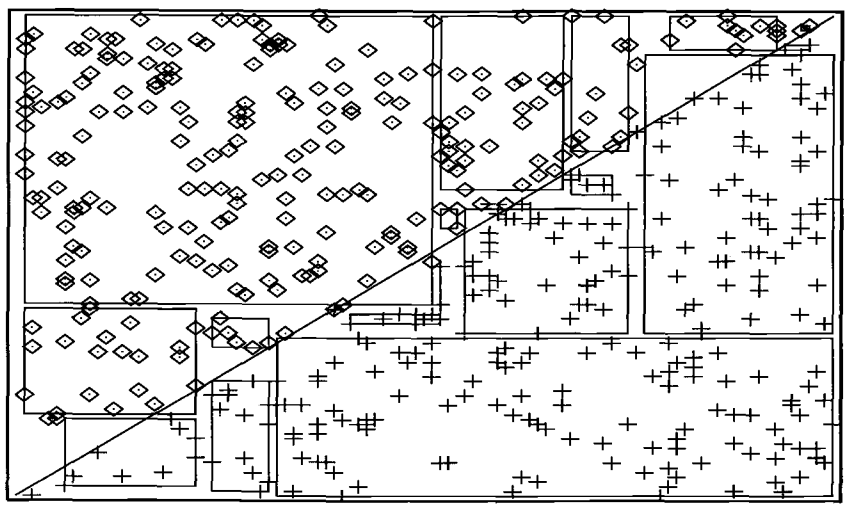


Figure 5.16. Rectangles generated by BNGE in the *diagonal* task. Plotted for comparison are also the original data points (+: class 0, ◇: class 1) and the decision boundary used to generate the data.

5.7 Feature Weights

The choice of the distance metric can significantly influence the performance of any distance-based machine learning algorithm in domains with continuous features (see also Section 4.3). Euclidean distance (L^2-norm) was used for NGE in all experiments reported in the previous sections. Note that the decision whether a query is inside or outside of a hyperrectangle is independent of the metric. On the other hand, the metric may heavily influence the number and shape of hyperrectangles constructed.

In all of the experiments we have conducted thus far, we have treated all features as equally important in computing the Euclidean distance to the nearest hyperrectangles (and nearest neighbors). However, many of the 11 domains involve noisy or completely random features. A way to improve performance of both NGE and kNN is to introduce some mechanism for learning which features are important and which unimportant (or noisy) features should be ignored in all distance computations.

Salzberg [Sal91, Section 3.3, last paragraph] describes a method for online learning of feature weights in NGE. Assume that a new example E is misclassified by an exemplar H. For each input feature f_i, if E_{f_i} matches H_{f_i}, the weight of f_i (w_{f_i}) is increased by multiplying it by $(1 + \Delta_f)$; if E_{f_i} does not match H_{f_i}, the weight w_{f_i} is decreased by multiplying it by $(1 - \Delta_f)$. Δ_f is the global feature-adjustment rate (usually set to 0.2). If E is classified correctly by H, then the feature weights are adjusted in the opposite direction.

There are two problems with this heuristic. First, consider tasks in which one class is much more frequent than another. In such tasks, new examples will tend to be classified correctly by chance, and the feature weights will change exponentially: features that always match will have weights of zero, and features that are random will receive infinite weight. Salzberg (personal communication & [Sal91, pseudo-code]) suggested adjusting feature weights only after both matches (to the nearest and second-nearest hyperrectangles) failed. We found empirically that with this policy, feature weights were adjusted quite infrequently. For example, in the Iris task, feature weights were adjusted for only 1% of the training examples. In Waveform-40,

Table 5.5. Percentage of (non-seed) examples covered by at least one hyperrectangle when NGE was initialized with 3 (25) seeds.

Domain	training		testing	
	3 seeds	25 seeds	3 seeds	25 seeds
Iris	64	29	87	57
Led-7 Display	79	81	93	91
Waveform-21	43	38	72	68
Waveform-40	23	20	51	45
Cleveland	80	62	97	91
Hungarian	83	52	98	81
Voting	97	93	100	100
Letter recognition	79	79	95	94

feature weights were adjusted for 7% of the training examples.

The second, more serious problem with the use of feature weights in NGE is that a high percentage of the test cases fall inside at least one hyperrectangle. The distance to the nearest hyperrectangle is zero in such a case, and feature weights would have no effect on the distance calculation (Table 5.5). This suggests that there are limits to the performance improvement that can be obtained by using feature weights with nested hyperrectangles.

In experiments with Salzberg's method for computing feature weights, we found that performance was almost always decreased (see below). Therefore, we considered another procedure for computing feature weights that has given promising results in other exemplar-based learning methods ([Bak91] and Section 4.3). This procedure is the Mutual Information procedure (Section 2.4.4).

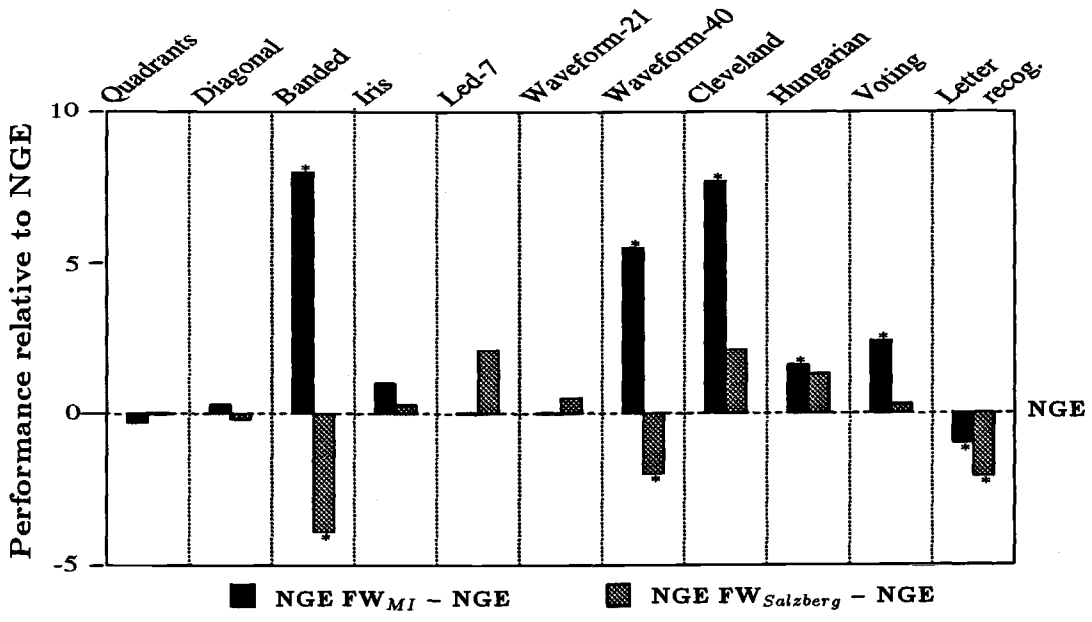


Figure 5.17. Performance of NGE FW$_{MI}$ and NGE FW$_{Salzberg}$ relative to NGE without feature weights. Shown are percentage point differences between NGE FW$_{MI}$ and NGE, and between NGE FW$_{Salzberg}$ and NGE. A * indicates significance of difference between NGE and its modifications. See Tables A.1 through A.5 for detailed numbers.

5.7.1 Experiments with Feature Weights

Figure 5.17 (and Tables A.1 through A.5) shows the effect of including feature weights and compares the two different procedures for computing the weights. Salzberg's method gave a statistically significant increase in performance in the Hungarian domain, a statistically significant decrease in the *banded* task, and had no significant effect in any of the other domains. The mutual information feature weights generally give slight, and statistically insignificant, improvements in domains without irrelevant features, but the improvements can be substantial in domains with irrelevant features. They give a statistically significant improvement in the Cleveland, Hungarian, Voting ($p < 0.01$), and Waveform-40 domains as well as in the *banded* task. A small ($p < 0.05$) decrease in performance is observed for mutual information feature weights in the Letter recognition domain. Mutual information feature weights had a similar positive effect on the performance of simple nearest neighbor and, to a

lesser extent, kNN (see Tables A.1 through A.5). The mutual information weights were very small for irrelevant inputs in all domains. Furthermore, feature weights did not differ substantially from one random partition of the data sets to another. In contrast, the weights computed by Salzberg's method differed substantially from one partition to another (varying by as much as a factor of 1000). Within a given training set/test set partition, the features were more or less equally weighted.

From the experiments in Section 5.7, we conclude that Salzberg's weight procedure has no significant impact on NGE's behavior in most domains and that the mutual information weight procedure performs well in domains that have a large number of irrelevant features. Furthermore, since the mutual information weight procedure is independent of the algorithm it is used for, it is a procedure that could be used effectively by many inductive learning algorithms to filter out irrelevant features.

5.8 Comparison of the Best Variants of NGE and kNN

We have now developed several modifications to NGE that uniformly improve its performance. The algorithm that best combines these is batch NGE with mutual information feature weights (BNGE FW_{MI}). The best corresponding version of the nearest neighbor algorithm is k nearest neighbors (with cross-validation to determine k) and mutual information feature weights (kNN FW_{MI}). Finally, the hybrid algorithm introduced in Section 5.6 with mutual information feature weights, KBNGE FW_{MI}, combines the advantages of both forementioned algorithms.

In this section, we compare these three algorithms to determine whether the modifications to NGE make it competitive with kNN. Figure 5.18 shows the results of this comparison. The main conclusion to draw is that BNGE FW_{MI} was significantly inferior to kNN FW_{MI} in 7 domains and significantly superior in only 2. The two domains are both domains where we know that the axis-parallel rectangle bias is appropriate. This shows that when such hyperrectangles are appropriate, BNGE FW_{MI} is able to exploit them. However, in domains where such rectangles are ev-

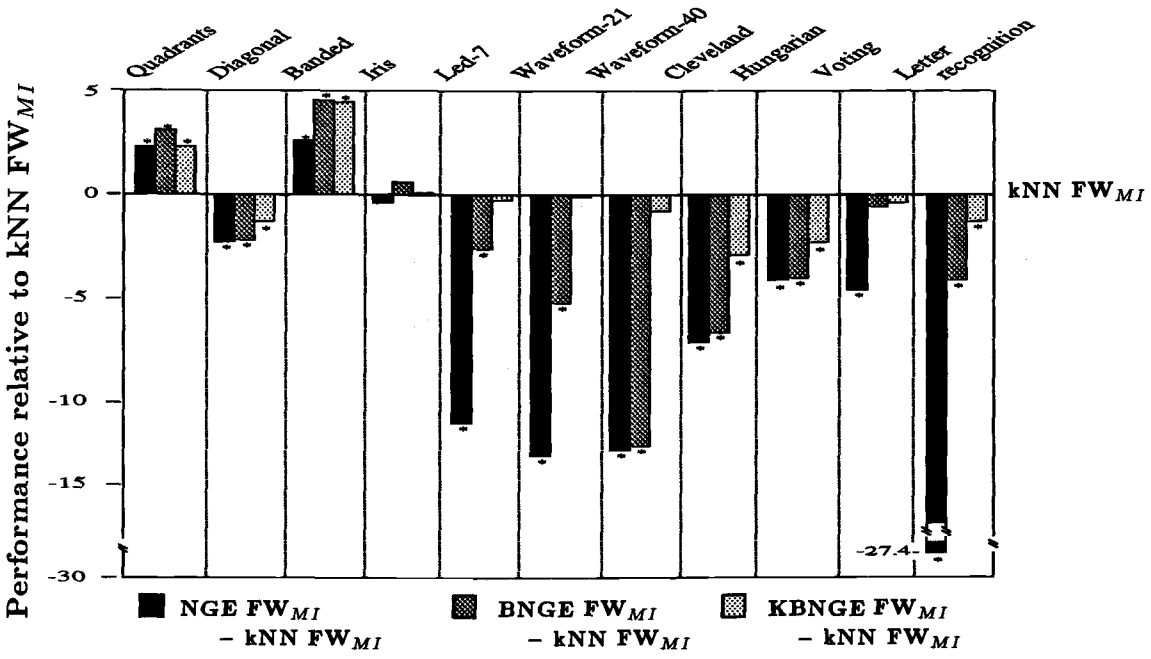


Figure 5.18. Performance of NGE FW_{MI}, BNGE FW_{MI}, and KBNGE FW_{MI} relative to kNN FW_{MI}. Shown are percentage point differences between NGE FW_{MI}, BNGE FW_{MI}, and KBNGE FW_{MI} and kNN FW_{MI}. These differences (*) were statistically significant. See Tables A.1 through A.5 for detailed numbers.

idently not appropriate, BNGE FW_{MI}'s performance suffers, while kNN FW_{MI} is robust. The KBNGE algorithm with feature weights achieved an accuracy comparable to kNN FW_{MI} in 7 of the 11 domains, but was significantly inferior to the kNN algorithm with feature weights in three domains.

5.9 Summary and Discussion

An extensive study of the NGE algorithm has been conducted. The basic algorithm and a number of modifications were evaluated in eleven domains. NGE was found to be quite sensitive to the number of starting seeds and to the order of presentation of the examples.

The performance of NGE was compared to the performance of the k-Nearest Neighbor algorithm and found to be substantially worse in several domains, even when cross-validation was applied to optimize the number of starting seeds in NGE.

Three hypotheses were introduced to explain this difference in performance: (a) nested rectangles provide a poor bias, (b) overlapping rectangles provide a poor bias, or (c) the incremental search algorithm of NGE needs improvement. Experimental modifications of NGE were made in order to test these hypotheses. Two versions of NGE that avoid nested rectangles (but permit overlapping rectangles) did not perform substantially better than NGE itself. However, an algorithm, called NONGE, which permits nested rectangles but avoids overlapping rectangles, performed uniformly better than NGE in all eleven domains (the improvement was statistically significant in 6 of the domains). A batch algorithm, BNGE, that implements a better search algorithm and does not allow nested or overlapping rectangles also performs uniformly better than NGE. It performs better than NONGE in 4 domains and worse in 2.

From these experiments, we conclude that overlapping rectangles are the primary source of difficulty for NGE and that BNGE was the best variant of NGE that we studied.

All versions of NGE were effective at compressing the data when compared to kNN. The amount of storage required by NGE and BNGE can be further reduced by employing a simple pruning technique (Section 5.5.1). This pruning technique removes exemplars that were never generalized from the classifier. This significant simplification of the classifier had no negative effect on the predictive accuracy of BNGE in any of the 11 domains tested (and in 10 of the 11 domains tested for NGE). A very compact representation of the training data is found after a classifier is constructed with BNGE and pruned. This representation can be used to do the following:

- Re-evaluate the representation. For example, we were able to determine that some of the input features were irrelevant in several domains through inspection of the hyperrectangles after training and pruning.

- Learn about the task. If only a few hyperrectangles are necessary to describe a task, then it can be said that it has a low level of noise. This small set of

hyperrectangles could then be used to construct a rule-based system to solve the task. If a large number of small hyperrectangles is necessary, then the task at hand is either extremely complex or the input representation is not powerful enough and should be modified.

- Assign levels of confidence to decisions. Queries that fall inside of hyperrectangles constructed by BNGE are significantly more likely to be classified correctly than queries outside of all hyperrectangles.

- Determine which regions of the input space are not adequately covered by training examples. This could prompt the experimenter either to collect more data or to clearly define which inputs can be processed by the system and which should be rejected. The ability for the user to easily interpret exemplars as prototypes of the task to be learned is a significant advantage of hyperrectangular based methods over such methods as kNN, neural networks, or decision trees.

A hybrid method, called KBNGE, that uses BNGE in areas that clearly belong to one output class and kNN otherwise was introduced and shown to have accuracy similar to kNN at improved classification speed in a large number of applications (Section 5.6). In the majority of the domains tested, over 70% of the test examples were classified by the hyperrectangular based part of KBNGE, thus making it significantly faster than kNN at classification time and enabling the system to justify most of its decisions in a manner that can be easily understood by the user. Note that the pruning technique used by KBNGE (un-generalized hyperrectangles are removed) influences the classification accuracy of KBNGE only for queries that perfectly match a given trivial hyperrectangle and only if $k \neq 1$. In all other cases, pruning only affects the speed of KBNGE.

A flaw of the current version of BNGE is that it constructs hyperrectangles only in those parts of the input space that contain no noisy data points. Future work will introduce noise tolerance into the BNGE algorithm by introducing a mechanism for accepting merges of hyperrectangles even if examples of other classes are

covered. A conceivable approach would be Omohundro's bottom-up model merging approach [Omo92], for example (see also Section 5.5).

The KBNGE algorithm exhibits classification accuracies comparable to the best known accuracies, it is fast in training and testing time, and it is easy to use. We believe KBNGE is an important tool to include in the set of commonly used machine learning algorithms.

We also studied whether the NGE algorithms could be improved by incorporating feature weights into the distance metric computed by the algorithms. The feature weight mechanism introduced by Salzberg [Sal91] was shown never to provide a significant improvement over the performance of NGE without feature weights. Indeed, it was significantly worse than simple NGE in three of the domains. On the other hand, a feature weight mechanism based on computing the mutual information between each feature and the output class was shown to be significantly better than NGE in five domains and significantly worse in only one. This mechanism is independent of NGE and can therefore be used as a pre-processing step for any inductive learning algorithm.

5.10 Conclusions

The data presented strongly support the conclusion that the NGE algorithm as described by Salzberg [Sal91] should be modified in a number of ways. Firstly, construction of overlapping hyperrectangles should be avoided. Secondly, if the entire training set is available at once and can be stored in memory, then the classifier should be trained in batch mode to eliminate computationally expensive cross-validation on the number of initial seeds. Thirdly, mutual information should be used to compute feature weights prior to running NGE.

With these modifications, NGE gives superior performance in domains where the axis-parallel hyperrectangle bias is appropriate. However, in other domains, NGE does not perform as well as kNN. Performance can be further improved when NGE is amended with a rejection criterion (KBNGE). The classifier will then reject

all queries that fall outside of all hyperrectangles. These queries should then be classified with the kNN classifier. This hybrid approach matches kNN's classification accuracy in most domains and is superior in domains where the axis-parallel bias is appropriate. Hence, if generalization performance and robustness are critical, kNN is the algorithm of choice. If, on the other hand, understandability and memory compression are important, then BNGE or KBNGE can be recommended as fast, easy-to-use inductive learning algorithms.

5.11 Related Work

5.11.1 Fuzzy Min-Max Neural Networks

Simpson [Sim92] introduced an incremental algorithm which is extremely similar to NGE called Fuzzy Min-Max Neural Networks. There are three main differences between NGE and a Fuzzy Min-Max Classifier (FMMC). (1) Hyperrectangles in FMMC are bounded in size. (2) The FMMC classifier always extends the nearest hyperrectangle that belongs to the same class as the training example to include that example. The extension of the nearest hyperrectangle is rejected when it would exceed a user-defined size. In which case, a new hyperrectangle is generated. (3) The FFMC classifier shrinks hyperrectangles to eliminate overlap of hyperrectangles from different classes.

We have seen in Section 5.4.2 that overlapping hyperrectangles from different classes are the primary cause for NGE's poor performance. Since FFMC avoids generation of overlapping hyperrectangles, it can be expected to generally perform better than NGE and comparable to BNGE. One potential problem with FFMC's hyperrectangle contraction process is that it cannot be modified to account for symbolic features. For continuous features, the hyperrectangle contraction process employed by FFMC may make it more robust than BNGE against noise in the data. However, the contraction process may cause erratic behavior in some cases. The reason for this being that FFMC first expands the nearest hyperrectangle to include the new training example, and then shrinks the new hyperrectangle and any other hyperrectangles

from other classes it overlaps with at approximately equal rates until all overlap is eliminated. This shrinking process does not take the number of training examples covered by each of the hyperrectangles into account. It may therefore cause FFMC to shrink a hyperrectangle that covers several hundred training examples because of a single noisy example. Furthermore, just as for BNGE, FFMC would suffer when irrelevant features are present in the data. These irrelevant features may fool FFMC into believing that two hyperrectangles do not overlap although they overlap in all relevant dimensions, or that two hyperrectangles from the same class are distinct from each other just because they have different values in at least one of the irrelevant features. I hypothesize that it may be more important to eliminate overlap on a dimension by dimension basis while allowing hyperrectangles to cover the entire range of other dimensions thereby effectively removing these dimensions.

Both, NGE and BNGE, can be easily modified to provide fuzzy membership outputs instead of the crisp classifications that are assigned to queries currently. The most obvious possibility would be to return the (possibly normalized) inverse of the distance of the query to the closest hyperrectangle from each class.

To summarize, FFMC is nearly identical to NGE despite the different vocabulary that was used to describe it. It should therefore suffer from similar shortcomings as NGE, and can be expected to perform generally inferior to KBNGE or kNN.

5.11.2 Fuzzy ARTMAP

Carpenter et al. [CGM+92] introduced a neural network architecture based on fuzzy logic and adaptive resonance theory (ART) neural networks. The category boxes used by fuzzy ARTMAP with complement coding are comparable to hyperrectangles. Hyperrectangles in fuzzy ARTMAP with complement coding grow monotonically during learning; their maximum size is bounded by a vigilance parameter. Carpenter et al. report a performance of 94.7% correct for Fuzzy ARTMAP in the Letter Recognition domain. That would be approximately 5 percentage points better then the performance that was observed for BNGE. However, this number is a performance that

was observed in one particular experiment for optimal settings for Fuzzy ARTMAP's free parameters as obtained from the test set. In a fair comparison, Fuzzy ARTMAP could be expected to perform comparably to BNGE. The main advantage of BNGE over Fuzzy ARTMAP is that BNGE has no free parameters while Fuzzy ARTMAP has at least two parameters that must be set by the user. Carpenter et al. [CGM+92] state in a short discussion comparing NGE, Fuzzy ARTMAP, and FMMC; that NGE allows hyperrectangles to shrink as well as to grow. This is not true for the present version of NGE described by Salzberg in [Sal91]. NGE's hyperrectangles can only grow in size. They further criticize that FMMC constructs only a single hyperrectangle for each output class. This statement is inconsistant with Simpson's description of the algorithm [Sim92].

Fuzzy ARTMAP may be able to outperform the other rectangular-based methods and even kNN if its free parameters are well chosen. However, it may be difficult to choose these values in a general setting.

Neither FMMC nor fuzzy ARTMAP use feature weights in the same sense as discussed in Section 5.7. They may therefore suffer more severely than BNGE FW_{MI} or kNN FW_{MI} when a large number of irrelevant features are present in the data.

Table 5.6. Percent accuracy (± standard error) of nearest neighbor (NN) and NGE on sythetic data sets. Numbers in parentheses show: number of seeds (first column, cv: leave-one-out cross-validation), average number of hyperrectangles constructed in all other columns.

	quadrants Task		*diagonal* Task		*banded* Task	
Method	Accuracy	Num Rect.	Accuracy	Num Rect.	Accuracy	Num Rect.
Nge (3)	86.8±4.0	(4.4±0.2)	75.6±1.7	(3.5±0.1)	87.4±0.5	(47.4±1.1)
Nge (5)	97.2±1.4	(7.0±0.3)	85.1±1.3	(6.1±0.2)	86.7±0.5	(48.8±1.0)
Nge (7)	97.8±0.7	(8.5±0.3)	86.3±1.3	(8.1±0.3)	86.2±0.5	(49.9±1.0)
Nge (10)	98.8±0.3	(11.6±0.3)	89.8±0.7	(11.1±0.3)	86.1±0.5	(49.8±1.0)
Nge (15)	98.0±0.4	(16.7±0.3)	92.4±0.5	(16.5±0.3)	85.6±0.5	(53.4±1.3)
Nge (20)	97.9±0.3	(21.8±0.3)	93.2±0.5	(21.6±0.3)	86.1±0.5	(57.0±1.1)
Nge (25)	97.8±0.3	(27.4±0.3)	93.7±0.6	(26.7±0.3)	85.3±0.5	(59.8±1.1)
Nge (cv)	99.3±0.2	(10.8±1.1)	94.1±0.5	(22.6±1.0)	86.9±0.5	(49.6±1.3)
NN	97.6±0.3		97.0±0.2		82.4±0.6	

Chapter 6
Summary, Conclusions and Future Work

In this final chapter, the results obtained from the experiments conducted in Chapters 3, 4 and 5 are summarized. These results are interpreted with regard to their impact on development, evaluation, and use of inductive machine learning algorithms in general and distance-based machine learning algorithms in particular. This dissertation is unique in that it attempts to explore the specific design decisions that control algorithm behavior and to provide a knowledge base for improved use of this family of algorithms in machine learning. Finally, some issues that deserve further investigation are discussed.

6.1 Summary

The objective of this dissertation was to develop a general framework for a family of inductive learning algorithms that we termed distance-based algorithms, and to present a detailed study of two members of this family, the nearest neighbor algorithm and the nearest-hyperrectangle algorithm. The specific results obtained from these detailed empirical evaluations are summarized below.

Six constructed data sets were employed in Chapter 3 to determine the circumstances under which the k-nearest neighbor algorithm achieves a classification accuracy superior to that of the first nearest neighbor algorithm. It was shown that in the noise-free case or for very small data sets, first nearest neighbor always performs better than kNN. In the presence of class or feature noise, values for k that are larger than 1 can lead to superior performance in certain data sets. These data sets must contain clusters of data from single classes that are wide enough along each dimension such that for larger values of k the majority of the newly included

neighbors belong to the same cluster. Hence, the optimal value for k depends not only on the amount of noise present in the data, but also on the shape of the decision boundaries within the data. A good distance metric should, therefore, rescale each input dimension to maximize for each data point the number of neighbors that belong to the same class as the data point. This also indicates that kNN is more likely to outperform NN when the distance metric is well chosen, while first nearest neighbor is the best we can do when the metric is poorly chosen.

The issue of how to estimate the optimal value for k from the training data was investigated in Chapter 4. It was determined that cross-validation on the value of k is necessary for best performance. A variety of cross-validation methods were compared. It was shown that while leave-one-out cross-validation on all possible values of k generally gives the best performance, comparable performance can be achieved if only one-fold cross-validation on a restricted number of values for k is conducted. These values for k should include a number of small values (k = 1,3,5,7,9,13,17) and only a few larger values (k = 27,35,41). These larger values were chosen arbitrarily due to the fact that kNN's performance was shown not to be sensitive to the exact choice of k when k is large. Several methods for choosing the value of k from local data alone were also suggested and examined. These methods gave good results on constructed data sets that were a combination of data sets with significantly varying attributes. The use of local nearest neighbor methods as studied in Chapter 4 can be recommended for data sets that exhibit substantially varying attributes for different subsets of their data points.

The experiments described in the second part of Chapter 4 support the conclusion that the votes of the k nearest neighbors should be weighted by their distances to the query when the query's classification is computed. This algorithm, termed kNN_{wv}, can be expected to give superior classification accuracy as compared to kNN with simple majority voting, particularly in domains where the optimal value of k is relatively small (< 25).

The potential of the method of Principal Component Analysis (PCA) for improving the performance of distance-based algorithms was studied in Section 4.3.4.

Principal Component Analysis offers three potential advantages for inductive learning algorithms: (1) It de-correlates input features. (2) It can be used to reduce the number of input dimensions. (3) It may represent the data in a manner that is more useful to inductive learning algorithms. It was shown in Section 4.3.4 that the method of Principal Component Analysis can be used to pre-process the input features of data sets with continuous features. Simple pre-processing with PCA has in general no significant effect on the performance of kNN. However, it was also shown that in many domains a relatively large number of features that resulted from PCA ("pca-features") can be removed. A general method for determining the number of pca-features that can be removed should be the topic of future research. The results presented here suggest that PCA may be used to remove a large number of input features to significantly reduce the time nearest neighbor algorithms require to classify queries. This may help to overcome one of the main disadvantages of nearest neighbor algorithms over neural networks, i.e. the fact that neural networks are in general substantially faster at classification time.

It is well established that the performance of any distance-based algorithm falls and rises with the choice of the distance function used to retrieve the most similar instances. The final part of Chapter 4 concerns the issue of how to find good feature weights to improve the classification accuracy of distance-based algorithms. Two methods were shown to compute good feature weights: (1) the method of mutual information, where the mutual information between each input feature and the outputs is computed and used as the feature weight, and (2) a method where feature weights are learned via gradient descent search. Both methods gave results superior to kNN without feature weights in a variety of domains. The largest improvements were observed in domains with many irrelevant features.

In Chapter 5, a method that combines data points into multidimensional rectangles (nested generalized exemplars, or NGE) is compared to the k-nearest neighbor algorithm and found to perform much worse in a variety of domains. The causes for this inferior performance were investigated, and it was found that performance of NGE can be substantially improved upon if generation of overlapping rectangles

from different classes is prohibited. A batch version of NGE, called BNGE, was developed. The BNGE algorithm constructs hyperrectangles only in areas of the input space that contain data points from a single class. It achieves substantial compression of the training data in all test domains and has classification accuracies that were either indistinguishable from or superior to NGE's. However, BNGE was superior to kNN only in tasks where decision boundaries were axis-parallel. Hence, a hybrid algorithm that combines the advantages of BNGE (fast classification, justification of classifications) with the accuracy of kNN was designed. The hybrid, called KBNGE, uses BNGE in areas where BNGE constructs hyperrectangles and kNN otherwise. The KBNGE algorithm was found to be substantially faster than kNN at classification time in all domains. Its generalization performance was comparable to kNN's in most cases, superior in domains where decision boundaries were axis-parallel, and inferior in some domains where BNGE over-generalized due to the fact that some of the input space did not contain any training examples.

6.2 Conclusions

The results of this dissertation have a number of practical implications on how distance-based algorithms should be modified. These modifications should enable any researcher (or user of one of these algorithms) to obtain good results in a variety of domains by choosing that combination of sub-biases of these algorithms that would seem most appropriate for each domain. Several new methods for determining some of the free parameters of any of these algorithms are proposed and shown to give good results in a number of domains. These domains are often characterized in detail so that the method's applicability to other domains can be predicted.

6.2.1 Recommendations Regarding Use of the k-Nearest Neighbor Algorithm

The k-nearest neighbor algorithm is an extremely powerful and versatile inductive learning algorithm. Its short training times and the fact that new training examples can simply be stored along with the old training examples makes this algorithm a

prime candidate in environments that continuously change, or when the user is unable or unwilling to invest a large amount of effort in setting parameters to obtain the best performance. Nearest neighbor algorithms are also good for probing a representation: researchers have often already acquired a number of data points and would like to know whether these data points (or the specific representation used) are sufficient to obtain the desired results. A kNN classifier is quickly and easily trained and can be used to answer just that question. If the kNN algorithm achieves results that are close to the desired results, then it may be worthwhile investing time to train and test other, more complicated classifiers.

We recommend the following procedure for training a k-nearest neighbor classifier: If features are symbolic, compute the mutual information feature weights for each input feature. If they are continuous, use the VSM training algorithm to learn the feature weights. Selection of the proper feature weights is the most important factor influencing the predictive accuracy of kNN. Parts of the training set should be set aside as test data for the 10 values of k listed above to determine the value of k that should be used to classify future test cases. The votes of the k nearest neighbors should have weights inversely proportional to their distances from the query when classifying the query.

6.2.2 Recommendations Regarding Use of the Nearest-Hyperrectangle Algorithm

We would recommend use of one of the modifications of NGE, as discussed in Chapter 5, in the following setting: Classification accuracy is not of primary concern, but an understanding of the decisions made by the classifier is very important. Whenever a specific rectangle is used to classify a query, then this rectangle can be translated into a human-readable rule. Such rules can then be employed not only to classify the query, but also to provide information with the classification, saying "this query was classified as belonging to class X since it satisfies the following conditions:..."

An additional advantage of methods such as BNGE is that they can achieve a substantial reduction in the amount of storage required compared to kNN.

A nearest hyperrectangle classifier should be constructed as follows: If trained incrementally, then a sufficient number of seeds (> 25, and at least one example from each class if the number of classes is known beforehand) should be stored without processing. During training, construction of overlapping rectangles from different classes should be avoided. If all training data are available before training, then the BNGE classifier should be constructed as described in Figure 5.9. If at all possible, then all training data should be retained and used to classify test examples that fall outside of all hyperrectangles. Previously seen training examples might also be used to split and reconstruct hyperrectangles during incremental training if it turns out that these hyperrectangles are too large.

6.3 Future Work

Some methods described in this dissertation represent only initial attempts aimed at improving the generalization accuracy and/or interpretability of results provided by distance-based algorithms. These methods, however, constitute a significant contribution to the research conducted in the fields of machine learning and pattern recognition. A complete understanding and robust implementation of these methods may assist researchers of distance-based algorithms in developing better algorithms, and users of distance-based algorithms in solving difficult real-world problems.

The mutual information procedure has improved the performance of kNN in nearly all domains. The largest improvements have been achieved in domains with irrelevant features or in domains where some features were substantially more important than others. However, in some domains with irrelevant features, the mutual information procedure has failed to assign weight 0 (or at least close to 0) to these irrelevant features. We suspect that this failure is mainly due to the procedure employed to estimate the probability distribution of continuous features. Silverman [Sil86] lists a number of methods for density estimation. Determination of

the most appropriate method from these techniques, and modification of the mutual information procedure currently used could improve the usefulness and general applicability of this method even further.

The training procedure employed by kNN FW_{VSM} to learn feature weights has two parameters that were kept constant in this research: the number of neighbors used to compute the gradient and the number of training epochs. An investigation of kNN FW_{VSM}'s sensitivity to the proper choice of these parameters, and the development of a method to choose the best values for these parameters should establish this method as an alternative method for learning the weights of features when the mutual information procedure is not applicable.

The mutual information feature weight estimation and the VSM feature weight learning techniques have been shown to improve the performance of the k-nearest and first-nearest neighbor algorithms quite substantially in domains with irrelevant features. An interesting issue is whether such improvement in classification accuracy could also be achieved when these methods are employed by other distance-based machine learning algorithms.

The experiments presented here have shown that many of the features output by principal component analysis (PCA) could be removed without a significant loss in classification accuracy. In some cases, removal even improved performance. A general method that reliably estimates the number of pca-features that can be removed without a loss in accuracy would prove extremely helpful, particularly in domains with very large numbers of input features such as speech recognition tasks. The primary purpose of this method in tasks with many input features would be to speed up the retrieval process during classification.

To this date, a greedy algorithm has been used to train the BNGE classifier. The algorithm is computationally very efficient, but it is possible that it does not construct the set of hyperrectangles that would in general result in the best performance. Implementation and evaluation of the BNGE Model Merging technique, as proposed in Section 5.5, can be used to determine whether the greedy procedure is sufficient for training BNGE.

The KBNGE algorithm had a classification accuracy that was inferior to kNN's in several domains. Apparently, these domains have large pockets in the input space that contain no training examples. The lower level of accuracy achieved by KBNGE in these domains must be attributed to the fact that BNGE over-generalizes by creating hyperrectangles that cover these empty pockets of the input space. This over-generalization should be avoided. A possible solution to this problem might be to limit the ratio of each hyperrectangle's size to the number of training examples it covers so that a hyperrectangle can only grow to large if it covers enough training examples.

The Euclidean L^2-norm was used in all experiments described in the fifth chapter. A different norm, such as the L^1-norm might cause rectangular based methods to generate different rectangles. A comparison of the rectangles constructed by BNGE, for example, when different norms are employed might further enhance our understanding of these algorithms and suggest further improvements.

This study has been limited to two basic algorithms, the nearest-neighbor algorithm and the nearest-hyperrectangle method. Many of the techniques investigated could also be applied to other distance-based algorithms such as Radial Basis Function Networks or Learning Vector Quantization. For example, it would be interesting to see whether principal component analysis could be used to improve the performance of RBF networks. These are interesting areas for future research.

BIBLIOGRAPHY

[Aha90] D.W. Aha. A study of instance-based algorithms for supervised learning tasks. Technical Report TR-90-42, University of California, Irvine, CA, November 1990.

[AKA91] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[Alm91] H.S. Almuallim. *Concept Coverage and its application to two learning tasks.* PhD thesis, Oregon State University, Corvallis, OR, 1991.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(2):312–342, 1988.

[Bak91] G. Bakiri. *Converting English Text to Speech: A Machine Learning Approach.* PhD thesis, Oregon State University, Corvallis, OR, 1991.

[BC89] E. Barnard and R.A. Cole. A neural-net training procedure based on conjugate-gradient optimization. Technical Report Rep. No. CSE 89-014, Oregon Graduate Institute, Beaverton, OR, 1989.

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[Bel61] R.E. Bellman. *Adaptive Control Processes: A guided Tour.* Princeton, NJ: Princeton Univerity Press, 1961.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and regression trees.* Belmont, CA: Wadsworth International Group, 1984.

[BJ78] T. Bailey and A.K. Jain. A note on distance-weighted k-nearest neighbor rules. *IEEE Tans. Syst., Man, Cybern.*, SMC-8:311–313, 1978.

[BL88] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[BV92] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[CGM+92] G.A. Carpenter, S. Grossberg, N. Markuzon, J.h. Reynolds, and D.B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, pages 698–713, 1992.

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[Che84] P.E. Cheng. Strong consistency of nearest neighbor regression function estimators. *Journal of Multivariate Analysis*, 15:63–72, 1984.

[CL93] E.I. Chang and R.P. Lippman. A boundary hunting radial basis function classifier which allocates centers constructively. In S.J. Hanson, J.D. Cowan, and Giles C.L., editors, *Neural Information Processing Systems 5*, pages 139–146. Morgan Kaufmann, San Mateo, CA, 1993.

[CMF90] R. Cole, Y. Muthusamy, and M. Fanty. The isolet spoken letter database. Technical Report No. CSE 90-004, Oregon Graduate Institute of Science and Technology, Department of Computer Science and Engineering, Beaverton, OR, 1990.

[Cov68] T. Cover. Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:50–55, 1968.

[CS93] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.

[CT91] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. John Wiley & Sons, INC., 1991.

[CU87] M.E. Connell and P.E. Utgoff. Learning to control a dynamic physical system. In *Proccedings of the Sixth National Conference on Artificial Intelligence*, pages 456–460, San Mateo, CA, 1987. Morgan Kaufmann.

[Das91] B.V. Dasarathy. *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, 1991.

[DH73] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[DHB90] T.G. Dietterich, H. Hild, and G. Bakiri. A comparative study of ID3 and backpropagation for english text-to-speech mapping. In *Proceedings of the 1990 Machine Learning Conference*, pages 24–31, 1990.

[DJS+89] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, K. Schmid, S. Sandhu, K. Guppy, S. Lee, and V. Froelicher. Rapid searches for complex patterns in biological molecules. *American Journal of Cardiology*, 64:304–310, 1989.

[Dud75] S.A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(4):325–327, 1975.

[FF82] K. Fukunaga and T.E. Flick. A parametrically-defined nearest neighbor distance measure. *Pattern recognition letters*, 1(1):3–5, 1982.

[FF84] K. Fukunaga and T.E. Flick. An optimal global nearest neighbor metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(3):314–318, 1984.

[FHJ51] E. Fix and J.L. Hodges, Jr. Discriminatory analysis, nonparametric discriminationm consistency properties. Technical Report 4, Randolph Field, TX: US Air Force, School of Aviation Medicine, 1951.

[Fis36] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7:179–188, 1936.

[Fis87] D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, pages 139–172, 1987.

[FJF77] J.H. Friedman, Bentley J.L., and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software.*, 3:209–226, 1977.

[FL90] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[Fla94] N.S. Flann. Recognition-based segmentation of on-line cursive handwriting. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Neural*

Information Processing Systems 6, pages 777–784. Morgan Kaufmann, San Mateo, CA, 1994.

[Flu88] B. Flury. *Common Principal Components and Related Multivariate Models*. John Wiley & Sons, 1988.

[FM88] D.H. Fisher and K.B. McKusick. An empirical comparison of ID3 and back-propagation. In *Procceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 788–793, 1988.

[Fog92] T.C. Fogarty. First nearest neighbor classification on frey and slate's letter recognition problem. *Machine Learning*, 9:387–388, 1992.

[Fri93] B. Fritzke. Kohonen feature maps and growing cell structures – a performance comparison. In S.J. Hanson, J.D. Cowan, and Giles C.L., editors, *Neural Information Processing Systems 5*, pages 123–130. Morgan Kaufmann, San Mateo, CA, 1993.

[FS91] P.W. Frey and D.J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6:161–182, 1991.

[FWD93] U.M. Fayyad, N. Weir, and S. Djorgovski. SKICAT: A machine learning system for automated cataloging of large scale sky surveys. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 112–119. Morgan Kaufmann, San Mateo, CA, 1993.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[Har68] P.E. Hart. The condensed nearest neighbor rule. *IEEE Trainsactions on Information Theory*, IT-14(3):515–516, 1968.

[Hau88] D. Haussler. Quantifying inductive bias: Ai learning algorithms and valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[HCG93] S.J. Hanson, J.D. Cowan, and C.L. Giles, editors. *Neural Information Processing Systems 5*. Morgan Kaufmann, San Mateo, CA, 1993. Part VIII: 649–722.

[Hol93] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[Hub85] P.J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

[IB91] Kononenko. I. and I. Bratko. Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6(1):67–80, 1991.

[JRLW87] K. Jabbour, J.F.V. Riveros, D. Landsbergen, and Meyer W. ALFA: Automated load forecasting assistant. In *Proceedings of the 1987 IEEE Power Engineering Society Summer Meeting.*, San Fransisco, CA., 1987. IEEE Press.

[KA87] D. Kibler and D.W. Aha. Learning representative exemplar of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 24–30. Irvine, CA: Morgan Kaufmann, 1987.

[KBC88] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks: Benchmarking studies. In *Proceedings of the International Joint Conference on Neural Networks*, pages I 61–68, 1988.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, third edition, 1989.

[Koh90a] T. Kohonen. Improved versions of learning vector quantization. In *Proceedings of the International Joint Conference on Neural Networks*, pages I 545–550, 1990.

[Koh90b] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, September 1990.

[Ksh72] A.M. Kshirsagar. *Multivariate Analysis.* Dekker, New York, 1972.

[LBD+90] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. In D.S. Touretzky, editor, *Neural Information Processing Systems 2*, pages 396–404. Morgan Kaufmann, San Mateo, CA, 1990.

[Lee91] Y. Lee. Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural Computation*, pages 440–449, 1991.

[LL90] Y. Lee and R.P. Lippmann. Practical characteristics of neural network and conventional pattern classifiers on artificial and speech problems. In *Neural Information Processing Systems 2*, pages 168–177, 1990.

[Low94] D.G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 1994. submitted to.

[MA91] P.M. Murphy and D.W. Aha. UCI repository of machine learning databases [machine-readable data repository]. Technical report, University of California, Department of Information and Computer Science. Irvine, CA, 1991.

[Mac67] J. MacQueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematics, Statistics, and Probability.*, pages 281–293, 1967.

[McG55] W.J. McGill. Multivariate information transmission. *IRE Trans. Inf. Theory*, 1:93–111, 1955.

[MD88] J. Moody and C. Darken. Learning with localized receptive fields. In *Proceedings of the 1988 Connectionists Models Summer School*, pages 133–143, 1988.

[MH90] J.P. Myles and D.J. Hand. The multi-class metric problem in nearest neighbor discrimination rules. *Pattern Recognition*, 23(11):1291–1297, 1990.

[Min89] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[Mit90] T.M. Mitchell. The need for biases in learning gneralizations. In J.W. Shavlik and T.G. Dietterich, editors, *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann, San Mateo, CA, 1990.

[MLT87] J.E.S. Macleod, A. Luk, and D.M. Titterington. A re-examination of the distance-weighted k-nearest-neighbor classification rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(4):689–696, 1987.

[MM94] O. Maron and A.W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Neural Information Processing Systems 6*. Morgan Kaufmann, San Mateo, CA, 1994.

[Mor76] D.F. Morrison. *Multivariate Statistical Methods.* McGraw-Hill, New York, 2nd edition, 1976.

[MSTG89] R. Mooney, J. Shavlik, G. Towell, and A. Gove. An experimental comparison of symbolic and connectionist learning algorithms. In *International Joint Conference on Artificial Intelligence*, 1989.

[NL91] K. Ng and R.P. Lippmann. A comparative study of the practical characteristics of neural network and conventional pattern classifiers. In *Neural Information Processing Systems 3*, pages 970–976, 1991.

[Omo89] S.M. Omohundro. Five balltree construction algorithms. Technical Report TR-89-063, International Computer Science Institute, Berkeley, CA, 1989.

[Omo92] S.M. Omohundro. Best-first model merging for dynamic learning and recognition. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Neural Information Processing Systems 4*, pages 958–965. Morgan Kaufmann, San Mateo, CA, 1992.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine, ser. 6*, 2:559–572, 1901.

[PG89] T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical Report MIT A.I. Memo No. 1140, MIT, July 1989.

[Pla90] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2), 1990.

[Pom93] D.A. Pomerleau. Input reconstruction reliability estimation. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Neural Information Processing Systems 5*, pages 279–286. Morgan Kaufmann, San Mateo, CA, 1993.

[PS93] J. Park and I.W. Sandberg. Approximation and radial-basis-function networks. *Neural Computation*, pages 305–316, 1993.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui88] J.R. Quinlan. An empirical comparison of genetic and decision-tree classifiers. In *Procceedings of the Fifth International Conference on Machine*

Learning, pages 135–141, 1988.

[Qui92] J.R. Quinlan. *C4.5: Programs for Empirical Learning.* San Mateo, CA: Morgan Kaufmann Publishers, INC., 1992.

[Ref92] A.N. Refenes. Constructive learning and its application to currency exchange rate predictions. In E. Turban and R. Trippi, editors, *Neural network applications in investment and finance services*, chapter 27. Chicago: Probus Publishing, 1992.

[Ren86] L. Rendell. A general framework for induction and a study of selective induction. *Machine Learning*, 1:177–226, 1986.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[RJ86] D.E. Rumelhart and J.L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. The MIT Press, 1986.

[Ros78] E. Rosch. Principles of categorization. In E. Rosch and B.B. Lloyd, editors, *Cognition and categorization.* Lawrence Erlbaum, Hillsdale, NJ, 1978.

[Sal91] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:277–309, 1991.

[Sch93a] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[Sch93b] C. Schaffer. Technical note: Selecting a classification method by cross-validation. *Machine Learning*, 13:135–144, 1993.

[SD86] C. Stanfill and Waltz. D. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[Seb62] G.S. Sebestyen. *Decision-making processes in pattern recognition.* New York, NY:The Macmillan Company, 1962.

[SF80] R.D. Short and K Fukunaga. A new nearest neighbor distance measure. In *Proceedings of the fifth international Conference on Pattern Recog-*

nition, pages 81–86, Los Alamitos, CA, 1980. IEEE Computer Society Press.

[SF81] R.D. Short and K Fukunaga. The optimal distance measure for nearest neighbor classification. *IEEE Transactions on Information Theory*, It–27(5):622–627, 1981.

[SF86] J.C. Schlimmer and D.H. Fisher. A case study of incremental concept induction. In *Procceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501, 1986.

[Sha48] C.E. Shannon. A mathematical theory of communication. *Bell. Syst. Techn. J.*, 27:379–423, 1948.

[Sil86] B.W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1986.

[Sim92] P.K. Simpson. Fuzzy min-max neural networks: 1. classification. *IEEE Transactions on Neural Networks*, 3:776–786, 1992.

[SL92] E. Singer and R.P. Lippmann. Improved hidden markov model speech recognition using radial basis function networks. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Neural Information Processing Systems 4*, pages 159–166. Morgan Kaufmann, San Mateo, CA, 1992.

[SLCD93] P. Simard, Y. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S.J. Hanson, J.D. Cowan, and C.L. Giles, editors, *Neural Information Processing Systems 5*, pages 50–58. Morgan Kaufmann, San Mateo, CA, 1993.

[SM81] E.E. Smith and D.L. Medin. *Categories and concepts.* Harvard University Press, Cambridge, MA, 1981.

[SR87] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[SS86] L. Sterling and E. Shapiro. *The Art of PROLOG.* MIT Press, Cambridge, MA, 1986.

[Sto74] M. Stone. Cross-validation choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[Tom76a] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC–6(6):448–452, 1976.

[Tom76b] I. Tomek. Two modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC–6(11):769–772, 1976.

[Utg88] P.E. Utgoff. An incremental ID3. In *Procceedings of the Fifth International Conference on Machine Learning*, pages 107–120, 1988.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[Ven94] J. Venn. *Symbolic Logic.* Chelsea Publishing Company, Bronx, New York, 2nd edition, 1894.

[WD92] D. Wettschereck and T.G. Dietterich. Improving the performance of radial basis function networks by learning center locations. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Neural Information Processing Systems 4*, pages 1133–1140. Morgan Kaufmann, San Mateo, CA, 1992.

[WD94a] D. Wettschereck and T.G. Dietterich. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *To appear in: Machine Learning*, 1994.

[WD94b] D. Wettschereck and T.G. Dietterich. Locally adaptive nearest neighbor algorithms. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Neural Information Processing Systems 6*, pages 184–191. Morgan Kaufmann, San Mateo, CA, 1994.

[Wei91] S.M. Weiss. Small sample error rate estimation for k-NN classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):285–289, 1991.

[Wet94] D. Wettschereck. A hybrid nearest-neighbor and nearest-hyperrectangle algorithm. In *7th European Conference on Machine Learning. LNAI-784*, pages 323–335. Springer Verlag, 1994.

[Wil72] D.L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2:408–421, 1972.

[WK89] S.M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *International Joint Conference on Artificial Intelligence*, 1989.

[WK91] S.M. Weiss and C.A. Kulikowski. *Computer Systems that learn.* Morgan Kaufmann Publishers, INC San Mateo California, 1991.

[Wol89] D. Wolpert. Constructing a generalizer superior to NETtalk via mathematical theory of generalization. *Neural Networks*, 3:445–452, 1989.

[XKO93] L. Xu, A. Krzyzak, and E. Ola. Rival penalized competitive learning for clustering analysis, RBF Net, and curve detection. *IEEE Transactions on Neural Networks*, 4(4):636–649, 1993.

APPENDIX

Appendix A
Summary of Data

This appendix contains a summary of all the data that has been obtained from the experiments described in this dissertation. Some of the numbers reported here may differ from numbers reported in the main body of the dissertation. This difference is in general due to the use of a different random seed and should not have any significant effect on the relative performance of the methods compared.

Table A.1. Percent accuracy (± standard error) on test set in the *quadrants*, *diagonal*, and *banded* tasks. Shown is mean performance over 25 repetitions, standard error (S.E.), and amount of memory (in percent of training data, if less than 100%) required by NGE (M column).

Method	Domain					
	Quadrants		*Diagonal*		*Banded*	
	Perf.±S.E.	M	Perf.±S.E.	M	Perf.±S.E.	M
NN	97.7±0.4		97.9±0.2		83.5±0.7	
kNN_{cv}	97.0±0.4		97.5±0.3		83.0±0.7	
kNN_{wv}	97.6±0.3		97.8±0.3		84.1±0.7	
NGE_{cv}	99.3±0.2	6	94.1±0.5	13	86.9±0.5	28
NGE_{3seeds}	86.8±4.0	3	75.6±1.7	2	87.4±0.5	27
NGE_{limit}	98.8±0.3	7	96.8±0.2		87.4±0.5	27
Greedy NGE_{cv}	98.1±0.2	15	94.5±0.4	23	83.4±0.6	60
F2+NOC_{cv}	99.2±0.2	6	93.3±0.6	12	93.4±0.3	6
$NONGE_{cv}$	99.5±0.2	5	94.7±0.4	18	86.9±0.5	28
OBNGE	97.3±0.5	2	91.4±0.5	12	93.3±0.2	7
BNGE	99.8±0.1	2	95.2±0.4	12	93.6±0.5	3
KBNGE	99.1±0.2		96.3±0.4		94.3±0.4	
NGE_{cv} FW_{MI}	99.0±0.2	5	94.4±0.5	13	94.9±0.3	10
NGE_{cv} FW_{S}	99.3±0.2	6	93.9±0.5	13	83.0±0.7	32
BNGE FW_{MI}	99.8±0.1	2	95.5±0.4	12	95.4±0.2	6
NN FW_{MI}	97.6±0.2		97.9±0.2		89.2±0.4	
kNN_{cv} FW_{MI}	97.0±0.3		97.7±0.3		88.8±0.4	
kNN_{wv} FW_{MI}	97.4±0.4		97.7±0.3		89.5±0.7	
kNN_{wv} FW_{VSM}	97.3±0.4		97.8±0.4		95.7±0.3	
KBNGE FW_{MI}	99.0±0.3		96.2±0.4		95.3±0.3	

Table A.2. Percent accuracy (± standard error) on test set in the *sinusoidal*, *radial*, and *gaussian* tasks. Shown is mean performance over 25 repetitions, and standard error (S.E.).

Method	Domain		
	Sinusoidal	*Radial*	*Gaussian*
NN	73.6±0.6	87.0±0.6	96.0±0.3
kNN_{cv}	73.2±0.7	86.9±0.6	97.4±0.2
kNN_{wv}	73.4±0.7	87.4±0.6	97.0±0.3
KBNGE	81.0±0.6	87.6±0.5	97.3±0.2
NN FW_{MI}	75.1±0.7	86.7±0.6	96.0±0.3
kNN_{cv} FW_{MI}	74.6±0.9	86.0±0.6	97.9±0.1
kNN_{wv} FW_{MI}	74.6±0.7	87.4±0.6	97.1±0.2
kNN_{wv} FW_{VSM}	83.5±0.7	87.3±0.7	97.1±0.2

Table A.3. Percent accuracy (± standard error) on test set in Iris and Led-7 Display domains. Shown is mean performance over 25 repetitions, standard error (S.E.), and amount of memory (in percent of training data, if less than 100%) required by NGE (M column).

Method	Domain					
	Iris		Led-7		Led-24	
	Perf.±S.E.	M	Perf.±S.E.	M	Perf.±S.E.	M
NN	95.2±0.4		70.5±0.5		48.5±0.7	
kNN_{cv}	95.6±0.5		72.3±0.6		69.4±0.6	
kNN_{wv}	95.6±0.4		73.0±0.6		70.2±0.6	
NGE_{cv}	93.7±0.6	31	59.8±1.0	67		
NGE_{3seeds}	92.0±0.6	9	56.0±1.2	61		
NGE_{limit}	94.4±0.6	50	64.2±0.7			
Greedy NGE_{cv}	94.7±0.5	31	62.7±0.9	87		
$F2{+}NOC_{cv}$	93.7±0.7	33	57.6±1.1	50		
$NONGE_{cv}$	94.0±0.6	33	62.7±0.9	71		
OBNGE	93.2±0.6	8	62.1±0.9	84		
BNGE	94.7±0.5	12	69.5±0.5	73		
KBNGE	95.8±0.4		72.1±0.6			
NGE_{cv} FW_{MI}	94.7±0.5	34	59.8±1.1	64		
NGE_{cv} FW_{S}	94.0±0.6	35	61.9±1.2	66		
BNGE FW_{MI}	94.7±0.6	11	68.6±0.4	74		
NN FW_{MI}	95.5±0.4		68.3±0.5		63.1±0.6	
kNN_{cv} FW_{MI}	95.6±0.4		70.1±0.6		73.2±0.7	
kNN_{wv} FW_{MI}	95.6±0.4		71.3±0.6		73.5±0.6	
kNN_{wv} FW_{VSM}	95.0±0.6		72.8±0.6		72.9±0.6	
KBNGE FW_{MI}	95.6±0.3		70.6±0.7			

Table A.4. Percent accuracy (± standard error) on test set in Waveform and Cleveland domains. Shown is mean performance over 25 repetitions, standard error (S.E.), and amount of memory (in percent of training data, if less than 100%) required by NGE (M column).

Method	Domain					
	Waveform-21		Waveform-40		Cleveland	
	Perf.±S.E.	M	Perf.±S.E.	M	Perf.±S.E.	M
NN	75.2±1.1		69.4±1.0		77.8±0.9	
kNN_{cv}	81.9±0.9		80.8±1.0		83.4±0.5	
kNN_{wv}	81.7±0.8		81.3±1.1		82.9±0.5	
NGE_{cv}	70.0±0.9	17	64.6±1.2	24	66.9±1.8	29
NGE_{3seeds}	69.3±0.8	9	64.2±1.2	14	55.0±1.2	5
NGE_{limit}	71.5±0.9	56	64.9±1.2	14	76.3±1.2	
Greedy NGE_{cv}	68.1±1.6	65	56.0±1.1	86	72.6±1.3	63
$F2{+}NOC_{cv}$	70.3±0.9	10	65.1±1.1	15	64.5±1.5	23
$NONGE_{cv}$	74.1±1.0	77	64.5±1.4	23	78.5±0.8	54
OBNGE	70.3±0.9	6	64.5±1.2	3	71.3±1.2	59
BNGE	69.8±1.5		66.7±1.4	91	77.5±1.1	71
KBNGE	81.3±0.8		80.4±1.0		80.6±0.7	
NGE_{cv} FW_{MI}	70.0±1.1	13	70.1±1.0	13	74.6±1.4	34
NGE_{cv} FW_{S}	70.5±0.8	16	62.6±1.2	20	69.0±1.3	33
BNGE FW_{MI}	71.0±1.1	97	69.2±1.2	90	77.5±0.9	50
NN FW_{MI}	76.3±0.7		78.3±0.8		77.9±1.0	
kNN_{cv} FW_{MI}	82.6±0.9		82.4±0.6		81.7±0.6	
kNN_{wv} FW_{MI}	82.2±0.9		83.4±0.8		83.7±0.6	
kNN_{wv} FW_{VSM}	81.9±0.9		81.0±1.1		82.7±0.7	

Table A.5. Percent accuracy (± standard error) on test set (Hungarian, Voting, and Letter Recognition domains). Shown is mean performance over 25 repetitions, standard error (S.E.), and amount of memory (in percent of training data, if less than 100%) required by NGE (M column).

Method	Domain					
	Hungarian		Voting		Letter Recognition	
	Perf.±S.E.	M	Perf.±S.E.	M	Perf.±S.E.	M
NN	75.9±0.8		86.9±0.8		95.8±0.1	
kNN_{cv}	82.0±1.0		92.0±0.4		95.8±0.1	
kNN_{wv}	82.5±1.0		92.4±0.4			
NGE_{cv}	76.5±0.9	30	88.4±1.2	16	70.2±0.4	41
NGE_{3seeds}	61.3±2.4	7	64.5±3.2	13	68.6±0.4	43
NGE_{limit}	79.3±0.8		84.8±1.1	22	87.4±0.2	
Greedy NGE_{cv}	78.6±0.9	49	88.8±1.2	34	70.1±0.4	63
$F2{+}NOC_{cv}$	73.7±1.1	21	88.4±1.2	14	63.9±0.4	24
$NONGE_{cv}$	79.3±0.7	43	88.3±1.6	22	88.0±0.2	39
OBNGE	71.6±1.4	40	88.8±2.4	25		
BNGE	76.7±1.0	36	93.2±0.5	47	89.1±0.1	39
KBNGE	80.6±0.9		94.1±0.4			
NN FW_{MI}	78.9±0.6		89.0±0.8		96.6±0.0	
kNN_{cv} FW_{MI}	82.2±0.9		95.4±0.4		96.6±0.0	
kNN_{wv} FW_{MI}	83.9±0.8		94.7±0.4			
kNN_{wv} FW_{VSM}	81.6±1.0		95.0±0.4			
NGE_{cv} FW_{MI}	78.1±0.6	28	90.8±0.8	15	69.2±0.4	35
NGE_{cv} FW_{S}	77.8±1.0	30	88.7±1.1	16	68.1±0.5	45
BNGE FW_{MI}	78.2±0.8	31	94.7±0.4	30	91.3±0.1	32

Table A.6. The performance of the k-nearest neighbor algorithm in synthetic tasks (noise-free) for different methods of determining the value of k. The subscript $k = i$ indicates that k was fixed at i, where *best* indicates the best average performance that was obtained by any single fixed value of k during 25 repetitions. Numbers in parentheses in the row denoted by $\text{kNN}_{k=best}$ indicate the range of k values that resulted in a performance within one standard error of the best performance. The subscript cv indicates cross-validation. The superscript to cv indicates the kind of potential candidates for k, and the subscript indicates the type of cross-validation. Significance of difference to $\text{kNN}_{k=1}$ ($\text{kNN}_{cv^{all}_{leave-1-out}}$) is indicated by * ($_{*}$).

Method	*Quadrants* ‡	*Diagonal* ‡	*Banded* ‡
$\text{kNN}_{k=1}$	97.2±0.3	98.2±0.2	86.9±0.6
$\text{kNN}_{k=3}$	97.0±0.2	98.1±0.2	85.6±0.5***
$\text{kNN}_{k=17}$	95.6±0.4$^{*****}_{****}$	97.4±0.2$^{****}_{***}$	77.4±0.6$^{*****}_{*****}$
$\text{kNN}_{k=31}$	95.2±0.4$^{*****}_{*****}$	97.3±0.3$^{****}_{**}$	65.3±0.7$^{*****}_{*****}$
$\text{kNN}_{k=best}$	97.2±0.3 (1-1)	98.2±0.2 (1-3)	86.9±0.6 (1-1)
$\text{kNN}_{cv^{odd}_{1-fold}}$	96.9±0.2	97.8±0.2**	86.1±0.6
$\text{kNN}_{cv^{all}_{leave-1-out}}$	97.0±0.2	98.1±0.3	86.2±0.6
$\text{kNN}_{cv^{odd}_{leave-1-out}}$	96.8±0.2*	98.2±0.2	86.2±0.6
$\text{kNN}_{cv^{1,3,17,31,47}_{leave-1-out}}$	96.8±0.2**	98.2±0.2	86.4±0.6
Method	*Sinusoidal* †	*Radial* ‡	*Gaussian* ‡
$\text{kNN}_{k=1}$	84.1±0.4	90.3±0.6$_{**}$	97.3±0.2$_{**}$
$\text{kNN}_{k=3}$	80.3±0.7$^{*****}_{*****}$	89.3±0.5**	97.5±0.2
$\text{kNN}_{k=17}$	62.8±0.8$^{*****}_{*****}$	83.9±0.9$^{*****}_{*****}$	97.9±0.2**
$\text{kNN}_{k=31}$	57.3±0.8$^{*****}_{*****}$	64.0±0.9$^{*****}_{*****}$	97.9±0.2
$\text{kNN}_{k=best}$	84.1±0.4 (1-1)	90.3±0.6 (1-1)$_{**}$	97.9±0.2 (5-43)**
$\text{kNN}_{cv^{odd}_{1-fold}}$	82.9±0.7$^{**}_{**}$	90.4±0.6$_{**}$	97.6±0.3**
$\text{kNN}_{cv^{all}_{leave-1-out}}$	84.1±0.4	89.3±0.5**	97.8±0.3**
$\text{kNN}_{cv^{odd}_{leave-1-out}}$	84.1±0.4	89.4±0.4*	97.8±0.2 **
$\text{kNN}_{cv^{1,3,17,31,47}_{leave-1-out}}$	84.1±0.4	89.7±0.6*	97.7±0.3**

† 850 training examples, 150 test examples

‡ 500 training examples, 150 test examples

$\star\star\star\star\star$ $p < 0.001$, $\star\star\star\star$ $p < 0.005$, $\star\star\star$ $p < 0.01$, $\star\star$ $p < 0.05$, $\star$ $p < 0.1$

Table A.7. The performance of the k-nearest neighbor algorithm in non-synthetic tasks for different methods of determining the value of k. The meaning of the terms used is the same as in Table A.6.

Method	Iris	Voting	Cleveland
$kNN_{k=1}$	95.2±0.4	$86.9\pm0.8_{*****}$	$77.8\pm0.9_{*****}$
$kNN_{k=3}$	95.3±0.4	$91.9\pm0.3^{*****}$	$82.2\pm0.6^{*****}_{*}$
$kNN_{k=17}$	95.5±0.5	$90.6\pm0.4^{*****}_{*****}$	$82.9\pm0.5^{*****}$
$kNN_{k=31}$	$93.2\pm0.8^{****}_{****}$	$89.8\pm0.4^{*****}_{*****}$	$83.7\pm0.5^{*****}$
$kNN_{k=best}$	96.3±0.4 (5-13)**	92.5 ± 0.4 (5-9)$^{*****}_{**}$	84.0±0.5 (19-57)*****
$kNN_{cv^{odd}_{1-fold}}$	95.2±0.4	$91.8\pm0.4^{*****}$	$82.5\pm0.8^{*****}$
$kNN_{cv^{all}_{leave-1-out}}$	95.6±0.5	$92.0\pm0.4^{*****}$	$83.4\pm0.5^{*****}$
$kNN_{cv^{odd}_{leave-1-out}}$	95.6±0.4	$92.2\pm0.4^{*****}_{*}$	$83.1\pm0.6^{*****}$
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	95.0±0.4	$92.3\pm0.4^{*****}_{***}$	$83.2\pm0.6^{*****}$
Method	Hungarian	Isolet	Led-7 Display
$kNN_{k=1}$	$75.9\pm0.8_{*****}$	$83.1\pm0.3_{*}$	$70.5\pm0.6_{*****}$
$kNN_{k=3}$	$79.2\pm0.7^{*****}_{*****}$	83.4±0.3	$72.4\pm0.5^{*****}$
$kNN_{k=17}$	$81.5\pm0.9^{*****}$	83.6±0.3	$64.8\pm1.1^{*****}_{*****}$
$kNN_{k=31}$	$82.0\pm0.8^{*****}$	$82.3\pm0.3^{**}_{*****}$	$59.0\pm1.3^{*****}_{*****}$
$kNN_{k=best}$	83.8±0.9 (37-71)$^{*****}_{***}$	84.0±0.4 (5-13)$^{***}_{**}$	72.6±0.5 (3-5)*****
$kNN_{cv^{odd}_{1-fold}}$	$80.9\pm0.9^{*****}$	83.4±0.3	$71.2\pm0.6^{*}_{*****}$
$kNN_{cv^{all}_{leave-1-out}}$	$82.0\pm1.0^{*****}$	83.6±0.2*	$72.3\pm0.6^{*****}$
$kNN_{cv^{odd}_{leave-1-out}}$	$82.1\pm1.0^{*****}$	83.6±0.2**	$72.4\pm0.6^{*****}$
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	$82.6\pm1.0^{*****}$	83.8±0.2**	$71.9\pm0.6^{*****}_{*}$

Table A.8. The performance of the k-nearest neighbor algorithm in non-synthetic tasks for different methods of determining the value of k. The meaning of the terms used is the same as in Table A.6.

Method	Led-24 Display	Waveform-21	Waveform-40
$kNN_{k=1}$	$48.5\pm0.7_{*****}$	$75.2\pm1.1_{*****}$	$69.4\pm1.0_{*****}$
$kNN_{k=3}$	$56.2\pm0.7^{*****}_{*****}$	$78.2\pm0.9^{****}_{*****}$	$73.2\pm0.7^{*****}_{*****}$
$kNN_{k=17}$	$67.1\pm0.7^{*****}_{*****}$	$81.8\pm0.8^{*****}$	$78.6\pm0.7^{*****}$
$kNN_{k=31}$	$68.9\pm0.6^{*****}$	$81.9\pm1.0^{*****}$	$79.7\pm0.8^{*****}$
$kNN_{k=best}$	69.9 ± 0.6 (43-67)$^{*****}_{**}$	82.1 ± 0.8 (17-55)*****	79.7 ± 0.8 (21-55)*****
$kNN_{cv^{odd}_{1-fold}}$	$68.3\pm0.7^{*****}_{**}$	$81.6\pm0.8^{*****}$	$79.6\pm1.0^{*****}$
$kNN_{cv^{all}_{leave-1-out}}$	$69.4\pm0.6^{*****}$	$81.9\pm0.9^{*****}$	$80.7\pm1.1^{*****}$
$kNN_{cv^{odd}_{leave-1-out}}$	$69.5\pm0.6^{*****}$	$81.8\pm0.9^{*****}$	$80.8\pm1.0^{*****}$
$kNN_{cv^{1,3,17,31,47}_{leave-1-out}}$	$69.6\pm0.5^{*****}$	$81.6\pm0.8^{*****}$	$80.4\pm1.1^{*****}$