UNIVERSITY OF CALIFORNIA
RIVERSIDE

Locally Adaptive Techniques for Pattern Classification

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in
Computer Science
by
Carlotta Domeniconi
August 2002

Dissertation Committee:
Prof. Dimitrios Gunopulos, Chairperson
Prof. Padhraic Smyth
Prof. Vassilis J. Tsotras

The Dissertation of Carlotta Domeniconi is approved:

Committee Chairperson

University of California, Riverside

ACKNOWLEDGMENTS

Each of the following faculties, researchers, student fellows, friends, and family members have contributed to this dissertation. Without each of them, this dissertation would have turned out differently, or may not exist at all.

First, I would like to thank Prof. Dimitrios Gunopulos for his support over the years, and the numerous and fruitful discussions that laid the foundation of the research presented here. In particular, I thank him for his knowledge passed on to me, and for the guidance that shaped this dissertation without constraining it.

I also thank Prof. Vassilis Tsotras for his encouragement and important feedback on my work. I thank Prof. Padhraic Smyth for his valuable comments and suggestions on this dissertation. I am grateful to Prof. Michael Jordan for his comments and useful suggestions in the early stage of my research. I thank Prof. Ping Liang for his support and enjoyable discussions during the first years of my graduate study.

I would like to thank Dr. Sheng Ma, Dr. Vittorio Castelli, Dr. Charles Perng, Dr. Ricardo Vilalta, and Dr. Irina Rish for many helpful and interesting discussions during my internship at IBM T. J. Watson Research Labs. I also thank Prof. Gianluca Bontempi for his comments on this dissertation, and Prof. Marek Chrobak for his feedback on my work.

A special thank goes to Prof. Jing Peng for uncountable discussions and brainstorms, from which I learned a great deal. He helped me stay focused throughout the development of this dissertation, and kept me cheerful and happy.

I thank the collegues and friends of the database lab and the department, for sharing the burden of the research through the years of my graduate study. Michalis Vlachos has made working in the database lab till morning hours more enjoyable. Thanks to Shihong, Fei, Eric, Lena, Jobin, Andria, and Sohail for the good times spent together. Marek deserves a special thank for keeping the motto *mens sana in corpore sano* alive among us, by organizing the famous (and strenuous) CS hikes (a.k.a. death marches).

I also thank Clare and Elijah, for the wonderful and cheerful conversations over tiramisù and coffee during our New York nights. From the east coast, they have been sharing this journey with me.

Last but not least, I would like to thank my parents Giorgio and Francesca Domeniconi, as well as my sisters Cosetta and Gioia, and my brother Luca. They always suppported me in pursuing my objectives. I own them my beliefs in education, and my willingness to learn, though it turned out not to be history or philosophy.

.

To the Curse of Dimensionality

ABSTRACT OF THE DISSERTATION

Locally Adaptive Techniques for Pattern Classification

by

Carlotta Domeniconi

Doctor of Philosophy, Graduate Program in Computer Science

University of California, Riverside, August, 2002

Prof. Dimitrios Gunopulos, Chairperson

Pattern classification faces a difficult challenge in finite settings and high dimensional spaces due to the curse of dimensionality. Nearest neighbor methods are especially sensitive to this problem. The need for large neighborhoods in high dimensional spaces is the cause of highly biased estimates. Due to the local nature of feature relevance, any chosen fixed metric violates the assumption of locally constant class posterior probabilities, and therefore fails in making correct predictions in different regions of the input space. In order to achieve accurate predictions, it becomes crucial to be able to estimate the different degrees of relevance that input features may have in various locations of the feature space.

This dissertation introduces novel approaches to computing local feature relevance for nearest neighbor methods that overcome limitations of previous techniques. It makes four specific contributions:

1. *ADAMENN algorithm*: A novel approach to computing local feature relevance for pattern classification. It uses the Chi-squared distance to design a flexible metric that approximates the theoretical infinite sample risk at the query point. No assumption is made on the probability distribution of data. It is formally shown that the measure of feature relevance derived by ADAMENN reduces the overall mean-squared estimation error.

2. *LFM-SVM algorithm*: A new local flexible metric technique based on support vector machines. This method overcomes the limitations of lazy learning approaches concerned with scalability and efficiency issues. It is shown that the weighting scheme performed by the LFM-SVM algorithm increases the margin of the solution provided in input by the SVM.

3. *GenProClus algorithm*: A novel algorithm that computes intra-cluster adaptive metrics for clustering. This algorithm represents an attempt to dodge the curse of dimensionality for clustering, and provides information to what features are relevant for each partition. It is shown that the algorithm converges to a local minimum of the associated error function.

4. *AdaBand algorithm*: A new locally adaptive technique to set the bandwidth parameters for kernel density estimation. This approach can serve as an efficient approximation of nearest neighbor methods. It is shown how this algorithm can be used to efficiently solve classification, clustering, and range query approximation problems.

The efficacy of the techniques presented is demonstrated through extensive experimental evaluations, using a variety of simulated and real world problems, such as texture recognition in images and letter classification.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Pattern Classification

The ability to classify patterns is certainly one of the key features of intelligent behavior, be it of humans or animals. This ability emerged with the biogenetic evolution for survival purposes, not only of individuals but also of entire species. An individual receives sensory information that must be processed to perceive, and ultimately act, possibly for orientation in the environment, distinction between edible and poisonous food, or detection of dangerous enemies.

Machine perception and classification as features of artificial systems serve similar but more constrained purposes. Machine classification aims at providing artificial systems with the ability to react to situations and signals coming from the environment to perform specific tasks. As such, pattern classification is a fundamental building block of any cognitive automata.

Pattern classification is a very general concept with numerous applications ranging from science, engineering, target marketing, medical diagnosis and electronic commerce to weather fore-

cast based on satellite imagery. A typical application of pattern classification is mass mailing for marketing. For example, credit card companies often mail solicitations to consumers. Naturally, they would like to target those consumers who are most likely to respond. Often, demographic information is available for those who have responded previously to such solicitations, and this information may be used in order to target the most likely respondents. Another application is electronic commerce of the new economy. E-commerce provides a rich environment to advance the state-of-the-art in classification because it demands effective means for text classification in order to make rapid product and market recommendations.

Recent developments in data mining have posed new challenges to pattern classification. Data mining is a knowledge discovery process whose aim is to discover unknown relationships and/or patterns from a large set of data, from which it is possible to predict future outcomes. As such, pattern classification becomes one of the key steps in an attempt to uncover the *hidden knowledge* within the data. The primary goal is usually predictive accuracy, with secondary goals being speed, ease of use, and interpretability of the resulting predictive model.

The term *pattern* is a word of our everyday vocabulary, and means something exhibiting some form of regularity, able to serve as a model representing a concept of what was observed. As a consequence, a pattern is never an isolated observation, but rather a collection of observations connected in time or space or both. A pattern exhibits, as a whole, a certain structure indicative of the underlying concept. The pattern classification task can then be seen as the task of inferring concepts from observations. Thus, designing a pattern classifier means defining a mapping from a measurement space into the space of possible meanings, that are viewed as finite and discrete target points.

From this perspective, it makes no difference what kind of observations are considered and to what kind of meanings they may be linked. The same approach can be used to recognize written text, spoken language, objects, or any other multidimensional signals as well. The selection of meaningful observations from a specific domain is a *feature extraction* process. From a theoretical viewpoint, the distinction between feature extraction and classification is arbitrary, but nevertheless useful. In general, the problem of feature extraction is much more domain dependent than the problem of classification.

This thesis focuses on classification. We are not concerned with the problem of feature extraction. We assume that a set of measurements is given, and we investigate the relevance of the given features for the classification task at hand. According to our point of view, no boundary exists between feature relevance estimation and classification, and the two processes are carried out together.

1.2 Statistical Approach

A characteristic of patterns in the context of classification is that every concept (or class) may have multiple representative points in the measurement space. For example, for the task of character recognition from their images, there exists a potentially unlimited plurality of ways of designing character images that correspond to the same character. Therefore, the very core of pattern classification is to cope with variability. The difficulty of the task depends on the degree to which the representatives of a class are allowed to vary and how they are distributed in the measurement space. This observation brings together two intrinsic components of the pattern classification task: the *statistical* component and the principle of *learning from examples*.

The problem of classification can be seen as one of partitioning the feature space into regions, one region for each category. Ideally, we would like to arrange this partitioning so that none of the decisions is ever wrong. This objective may not be achievable for two reasons. The distributions of points of different classes in the measurement space overlap, so that we cannot reliably separate one class from the other. Moreover, even if we are able to find a rule that does a good job of separating the examples, we have no guarantee that it will perform as well on new points. In other words, that rule may not *generalize* well on data never seen before. It would certainly be safer to consider more points, and check how many of those are correctly classified by the rule. This suggests that we should look for a classification procedure that aims at minimizing the *probability of error*. The problem of classification becomes then a problem in statistical decision theory.

1.3 Challenges

While pattern classification has shown promise in many areas of practical significance, it faces difficult challenges from real world problems, of which the most pronounced is Bellman's *curse of dimensionality* [Bel61]. It states the fact that the sample size required to perform accurate prediction in problems with high dimensionality is beyond feasibility. This is because in high dimensional spaces data become extremely sparse and are apart from each other. As a result, severe bias that affects any estimation process can be introduced in a high dimensional feature space with finite samples.

Consider, for example, the rule that classifies a new data point with the label of its closest training point in the measurement space (*1-Nearest Neighbor rule*). Suppose each instance is described by 20 attributes, but only three of them are relevant to classifying a given instance. In

this case, two points that have identical values for the three relevant attributes may nevertheless be distant from one another in the 20-dimensional input space. As a result, the similarity metric that uses all 20 attributes will be misleading, since the distance between neighbors will be dominated by the large number of irrelevant features. This shows the effect of the curse of dimensionality phenomenon, that is, in high dimensional spaces distances between points within the same class or between different classes may be similar. This fact leads to highly biased estimates. Nearest neighbor approaches are especially sensitive to this problem.

In many practical applications things are often further complicated. In the previous example, the three relevant attributes for the classification task at hand may be dependent on the location of the query point, i.e. the point to be classified, in the feature space. Some features may be relevant within a specific region, while other features may be more relevant in a different region.

These observations have two important implications. Distance computation does not vary with equal strength or in the same proportion in all directions in the feature space emanating from the input query. Moreover, the value of such strength for a specific feature may vary from location to location in the feature space. Capturing such information, therefore, is of great importance to any classification procedure in high dimensional settings.

We emphasize that the curse of dimensionality is not confined to classification. It affects any estimation process in a high dimensional feature space with finite examples. Thus, clustering equally suffers from the same problem. The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. It is not meaningful to look for clusters in high dimensional spaces as the average density of points anywhere in input space is

likely to be low. As a consequence, distance functions that equally use all input features may be ineffective.

In this thesis we investigate new classification and clustering techniques to mitigate the curse of dimensionality and reduce bias. This issue has both a theoretical and practical relevance, since many applications could benefit from such an improvement in prediction error. We lay a theoretical foundation upon which to build adaptive metric machines for classification. We then describe effective procedures for operating such a machine, and later extend the setting for clustering. We show results demonstrating the efficacy of our methods using a variety of simulated and real world problems, such as texture recognition in images and letter classification.

1.4 Dissertation Overview

The goal of this chapter has been to set up the appropriate context within which this dissertation is to be developed. An introduction to the problem of pattern classification has been presented, angled toward its intrinsic statistical nature, and the learning from examples paradigm. We have also emphasized the reasons why pattern classification and clustering are difficult and complex tasks, expecially when the examples lie in a high dimensional feature space.

The contributions of this dissertation can be briefly summarized as follows:

1. *ADAMENN algorithm*: A novel approach to computing local feature relevance for pattern classification;

2. *LFM-SVM algorithm*: A new local flexible metric technique based on support vector machines;

3. *GenProClus algorithm*: A novel algorithm that computes intra-cluster adaptive metrics for clustering;

4. *AdaBand algorithm*: A new locally adaptive technique to set the bandwidth parameters for kernel density estimation, and its applications to efficiently solve classification, clustering, and range query approximation problems.

The content of the chapters of this dissertation is organized as follows:

Chapter 2 introduces background work in pattern classification, focusing on the *supervised learning* paradigm. It begins by discussing the concepts of Bayes decision theory, that provides a fundamental statistical approach to the problem of pattern classification. It then goes on to describe different possible approaches to estimating the posterior probability values. Particular emphasis is given to nearest neighbor methods, since they represent the core of this thesis. Finally, it discusses the main concepts and properties of *support vector machines*, that will be used in later chapters.

Chapter 3 describes the problem caused by finite settings in high dimensional spaces, highlighting the importance of estimating the different degree of relevance that input features may have in various locations in the feature space. It discusses previous work in the literature on flexible metric computation along with their limitations.

Chapter 4 presents a novel approach (ADAMENN) to computing local feature relevance. It first lays a theoretical foundation upon which to build an adaptive metric machine, and then describes an effective procedure for operating such a machine. It discusses the question of convergence of our method, and proves that our measure of feature relevance reduces the overall mean-squared

estimation error.

Chapter 5 presents an extensive experimental evaluation comparing ADAMENN with well-known techniques in the literature, using both simulated and real data. The results obtained strengthen the theoretical properties of our technique discussed in the previous chapter.

Chapter 6 addresses the limitations of lazy learning approaches concerned with scalability and efficiency issues. It presents a new locally flexible metric technique (LFM-SVM) based on support vector machines that overcomes some of the drawbacks of ADAMENN. It proves that the weighting scheme performed by our technique increases the margin of the solution provided by the support vector machine.

Chapter 7 presents an extensive experimental evaluation comparing LFM-SVM with a variety of classification approaches, including SVMs and ADAMENN. Both simulated data with an increasing number of noisy predictors and real data are used.

Chapter 8 introduces a novel algorithm (GenProClus) that computes intra-cluster adaptive metrics for clustering. The algorithm discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This algorithm represents an attempt to dodge Bellman's curse of dimensionality for clustering problems. The technique provides information to what features are relevant for each partition. The chapter presents a proof of convergence of GenProClus to a local minimum of the associated error function.

Chapter 9 presents an experimental evaluation comparing GenProClus, EM, and K-means algorithms, and using a variety of simulated data sets. The superior accuracy achieved by GenProClus in high dimensional spaces provides evidence of the feasibility of the approach. Experimental

results also show a perfect corresponence between the weight values of each cluster and local correlations of data.

Chapter 10 proposes a new locally adaptive technique to address the problem of setting the bandwidth parameters for kernel density estimation. The technique is efficient and can be performed in only two data passes. It is also shown how to apply the method to efficiently solve range query approximation, classification and clustering problems in very large data sets.

Chapter 11 validates the efficiency and accuracy of locally adapting the bandwidth parameters for kernel density estimation, for both range query approximation and classification problems.

Finally, Chapter 12 summarizes the work presented in the previous chapters. It contains a brief description of the new algorithms and concepts introduced in this dissertation. A short description of important directions for future research concludes the dissertation.

Chapter 2

Background

Bayes decision theory provides a fundamental statistical approach to the problem of pattern classification. It is based on the assumption that the decision problem is posed in probabilistic terms, and that all relevant probability values are known. In the following we introduce *supervised learning* approaches to estimating the posterior probability values, in an attempt to come as closer as possible to the optimum Bayesian classifier. The chapter concludes with a discussion on the main concepts and properties of support vector machine classifiers.

2.1 Classification

In a classification problem an observation is characterized by q feature measurements $\mathbf{x} = (x_1, \cdots, x_q) \in \Re^q$ and is presumed to be a member of one of J classes, $L_j, j = 1, \ldots, J$. The particular group is unknown, and the goal is to assign the given object to the correct group using its measured values $\mathbf{x}$.

Let $\lambda(j|k)$ be the loss incurred by assigning $\mathbf{x}$ to the jth group L_j, when it is actually a member of the kth group L_k [DH73]. By denoting $P(k|\mathbf{x})$ the probability that $\mathbf{x}$ is a member of the kth class given the particular set of measurements $\mathbf{x}$, the *expected loss* associated with class j is

$$R(j|\mathbf{x}) = \sum_{k=1}^{J} \lambda(j|k) P(k|\mathbf{x}), \tag{2.1}$$

which is also known as the *conditional risk*. Whenever we encounter a particular observation $\mathbf{x}$, we can minimize the expected loss by selecting the class that minimizes the conditional risk,

$$j^* = \arg \min_{1 \leq j \leq J} R(j|\mathbf{x}). \tag{2.2}$$

A loss function of particular interest is the so-called *symmetrical* or *zero-one* loss function,

$$\lambda(j|k) = \begin{cases} 0 & \text{if } j = k \\ 1 & \text{if } j \neq k \end{cases} \tag{2.3}$$

where $j, k = 1, \cdots, J$. This loss function assigns no loss to a correct decision and a unit loss to any error. Thus, all errors are equally costly. The risk corresponding to this loss function is the average probability of error, or the *error rate*, since the conditional risk is

$$\begin{aligned} R(j|\mathbf{x}) &= \sum_{k=1}^{J} \lambda(j|k) P(k|\mathbf{x}) \\ &= \sum_{k \neq j} P(k|\mathbf{x}) \\ &= 1 - P(j|\mathbf{x}), \end{aligned} \tag{2.4}$$

and $P(j|\mathbf{x})$ is the conditional probability that class j is correct. Therefore, equation (2.2) reduces to

$$j^* = \arg \max_{1 \leq j \leq J} P(j|\mathbf{x}) \tag{2.5}$$

if all misclassifications are considered equally costly. Equations (2.2) and (2.5) are known as the Bayes decision rules and their associated error rate is the minimum achievable. Thus, to minimize the average probability of error, we should select the class j that *maximizes* the posterior probability $P(j|\mathbf{x})$. In other words, for *minimum error rate*:

$$Decide \;\; j \;\; if \;\; P(j|\mathbf{x}) > P(k|\mathbf{x}) \;\; for\, all\, k \neq j.$$

In order to apply the Bayes decision rule the true conditional probabilities $\{P(j|\mathbf{x})\}_{j=1}^{J}$ must be known for every query point $\mathbf{x} \in \Re^q$ at which class predictions are to be made. Unfortunately, this is seldom the case, and the conditional probabilities must be estimated.

Bayes decision theory provides a fundamental statistical approach to the problem of pattern classification. It is based on the assumption that the decision problem is posed in probabilistic terms, and that all relevant probability values are known. In the following we introduce *supervised learning* approaches to estimating the posterior probability values, in an attempt to come as closer as possible to the optimum Bayesian classifier.

We are given J classes and N training observations. The training observations consist of q feature measurements $\mathbf{x} = (x_1, \cdots, x_q) \in \Re^q$ and the known class labels, $j = 1, \ldots, J$:

$$\{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \tag{2.6}$$

Here $y_n \in \{1, \ldots, J\}$. The goal is again to predict the class label of a given query $\mathbf{x}_0$. The training data (2.6) are used to obtain the estimates

$$\{\hat{P}(j|\mathbf{x}_0)\}_{j=1}^{J} \tag{2.7}$$

which are then used in (2.1) and (2.2) to get the class assignment estimate

$$\hat{j}(\mathbf{x}_0) = \arg \min_{1 \le j \le J} \sum_{k=1}^{J} \lambda(j|k) \hat{P}(k|\mathbf{x}_0), \tag{2.8}$$

or the corresponding analog for (2.5):

$$\hat{j}(\mathbf{x}_0) = \arg \max_{1 \le j \le J} \hat{P}(j|\mathbf{x}_0). \tag{2.9}$$

2.2 Density Estimation

One approach to estimating the posterior class conditional probabilities is based on density estimation via Bayes theorem

$$P(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)P(j)}{\sum_{k=1}^{J} p(\mathbf{x}|k)P(k)}. \tag{2.10}$$

Here $p(\mathbf{x}|j)$ is the probability density function of $\mathbf{x}$ given class j, and $P(j)$ is the probability that reflects our *prior* knowledge of observing an object of class j in the absence of a set of measurement values $\mathbf{x}$. Thus, Bayes rule shows how observing the values $\mathbf{x}$ changes the a priori probability $P(j)$ to the posterior probability $P(j|\mathbf{x})$.

With this approach, the training data in each class j, for $j = 1, \cdots, J$, are used separately to estimate the corresponding probability density function $p(\mathbf{x}|j)$ over the measurement space. The prior probabilities $P(j)$ are either known in advance through some knowledge of the nature of the problem under study, or are estimated as the proportions of each class in the training data set. These estimates are then used to derive the posterior class conditional probabilities (2.7) through (2.10).

2.2.1 Discriminant Analysis

An example of the density estimation approach is *discriminant analysis* [Mcl92]. In this case, the class conditional density functions are approximated by Gaussian distributions, and the training data for each class are used to estimate the mean vector and the covariance matrix for that class. The goal of discriminant analysis is to find *discriminant functions* in feature space that achieve the lowest error rate to the extent that the normal assumption is correct. Thus, a classifier is defined in terms of a set of discriminant functions $g_j(\mathbf{x}), j = 1, \cdots, J$. The classifier assigns a feature vector $\mathbf{x}$ to class j if

$$g_j(\mathbf{x}) > g_k(\mathbf{x}) \quad \forall\, k \neq j.$$

In words, the classifier selects the class corresponding to the largest discriminant. A Bayes classifier can easily be represented in this way. We can define $g_j(\mathbf{x}) = -R(j|\mathbf{x})$, since the maximum discriminant function will then correspond to the minimum conditional risk. For the minimum error rate case, things could be further simplified by taking $g_j(\mathbf{x}) = P(j|\mathbf{x})$, so that the maximum discriminant function corresponds to the maximum posterior probability. The discriminant functions can be written in a variety of forms, all resulting in the same classification result. In general, by replacing every $g_j(\mathbf{x})$ with $f(g_j(\mathbf{x}))$, where f is a monotonically increasing function, resulting classification is unchanged. In particular, for minimum error rate classification, it is convenient to consider

$$g_j(\mathbf{x}) = \log p(\mathbf{x}|j) + \log P(j) \tag{2.11}$$

as a discriminant function, which is equivalent to having $g_j(\mathbf{x}) = P(j|\mathbf{x})$. The expression in (2.11) can be evaluated if we assume that the densities $p(\mathbf{x}|j)$ are multivariate normal. Let $p(\mathbf{x}|j) \sim$

$N(\boldsymbol{\mu}_j, \Sigma_j)$, where

$$N(\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{q/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})]. \tag{2.12}$$

Then,

$$g_j(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^t \Sigma_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) - \frac{q}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_j| + \log P(j). \tag{2.13}$$

A simple case arises when the covariance matrices for all classes are identical. Since $|\Sigma_j|$ in (2.13) becomes independent of j, it can be ignored, along with the constant $(q/2)\log 2\pi$, resulting in the discriminat functions

$$g_j(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) + \log P(j). \tag{2.14}$$

If the a priori probabilities $P(j)$ are the same for all J classes, then the term $\log P(j)$ can be ignored. In this case, the decision rule can be stated very simply: To classify a feature vector $\mathbf{x}$, measure the distance $(\mathbf{x}-\boldsymbol{\mu}_j)^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)$ (known as the *squared Mahalanobis distance*) from $\mathbf{x}$ to each of the J mean vectors, and assign $\mathbf{x}$ to the category of the nearest mean. Unequal a priori probabilities bias the decision towards the a priori more likely class.

By expanding the term $(\mathbf{x}-\boldsymbol{\mu}_j)^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)$ it can be shown that the quadratic term $\mathbf{x}^t \Sigma^{-1}\mathbf{x}$ is independent of j. By deleting it, we obtain linear discriminant functions:

$$g_j(\mathbf{x}) = \mathbf{w}_j^t \mathbf{x} + w_{j0}, \tag{2.15}$$

where

$$\mathbf{w}_j = \Sigma^{-1}\boldsymbol{\mu}_j \tag{2.16}$$

and

$$w_{j0} = -\frac{1}{2}\boldsymbol{\mu}_j^t \Sigma^{-1}\boldsymbol{\mu}_j + \log P(j). \tag{2.17}$$

Thus, the resulting decision boundaries are hyperplanes. They define the optimal decision rule when the normal and equal covariance assumptions are satisfied by the actual underlying distributions of the data.

2.2.2 Naive Bayesian Classifier

The so-called "naive" Bayesian classifier makes the assumption that the values of the attributes of an example are independent given the class of the example. As a consequence, the joint density function in (2.10) can be written as the product of the marginal densities of each attribute, making the use of Bayes theorem much simpler. Although this assumption is almost always violated in practice, naive Bayesian learning is remarkably effective in practice [DP96].

2.3 Regression

The density estimation approach to classification computes posterior probabilities by estimating the probability density functions conditioned on the value of the class label. A second category of supervised learning techniques directly estimate the probabilities $\{P(j|\mathbf{x})\}_{j=1}^{J}$. At a given query point $\mathbf{x}$ the class label y is assumed to be a random variable from a multinomial distribution with probabilities $\{P(j|\mathbf{x})\}_{j=1}^{J}$. Each possible value for y at $\mathbf{x}$ is characterized by an additional output variable g_j such that

$$g_j|\mathbf{x} = \begin{cases} 1 & \text{if } y = j \\ 0 & \text{otherwise} \end{cases}$$

where $j = 1, \cdots, J$. $g_j|\mathbf{x}$ represents the characteristic function of class j at $\mathbf{x}$ and gives rise to J separate training sets, one for each class j:

$$\{\mathbf{x}_n, g_j\}_{n=1}^{N}, \quad j = 1, \cdots, J. \tag{2.18}$$

Then

$$f_j(\mathbf{x}) \stackrel{\text{def}}{=} P(j|\mathbf{x}) = P(g_j = 1|\mathbf{x}) = E[g_j|\mathbf{x}]. \tag{2.19}$$

The posterior probabilities are the target functions $f_j(\mathbf{x})$ to be estimated by using the corresponding training samples (2.18)

$$f_j(\mathbf{x}) = \arg \min_f E[(g_j - f)^2|\mathbf{x}], \tag{2.20}$$

where $j = 1, \cdots, J$, so that they become the solutions of a set of least-squares problems. It is evident that, since they represent probabilities, the target functions $\{f_j(\mathbf{x})\}_{j=1}^{J}$ satisfy the constraints

$$0 \leq f_j(\mathbf{x}) \leq 1, \quad and \quad \sum_{j=1}^{J} f_j(\mathbf{x}) = 1.$$

Many of the techniques developed in machine learning and pattern recognition apply more or less closely this regression paradigm to the classification problem. We elaborate more on some specific methods such as *neural networks*, *decision trees*, *K-nearest-neighbors*, and *support vector machines*.

2.3.1 Neural Networks: Generalized Linear Models (GLIM)

We describe how a two class classification problem can be modeled by using a neural network representation [Bis95]. The interpretation of the output of the resulting neural network model as posterior probabilities has a statistical foundation in Bayes theory [Jor95].

Suppose the two classes have labels 0 and 1. For a given point $\mathbf{x}$, its class label y is assumed to be a random variable from a Bernoulli distribution with probabilities $\{P(j|\mathbf{x})\}_{j=0}^{1}$:

$$p(y;\mu) = \mu^y(1-\mu)^{1-y} \tag{2.21}$$

where $\mu = P(1|\mathbf{x})$ is the posterior probability associated with class 1, $P(0|\mathbf{x}) = (1-\mu)$, and $y \in \{0,1\}$. Equation (2.21) represents the Bernoulli mass function.

Our goal is to estimate the posterior probability value μ. We are given a training set $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Here $y_i = \{0,1\}$. The learning system needs to infer the probability value μ by the given pairs $(\mathbf{x}_i, y_i)$ in X. We can write $\mu = P(1|\mathbf{x}) = f(\mathbf{w}^t\mathbf{x})$, which means: given $\mathbf{x}$ as input, the corresponding probability of observing the output $y = 1$ is $P(1|\mathbf{x}) = f(\mathbf{w}^t\mathbf{x})$. Here $\mathbf{x}$ undergoes a linear transformation, whose result is then mapped to the interval $[0,1]$ by a non linear "squashing" function f. The parameters (*weights*) of the linear transformation $\mathbf{w} = (w_1, \cdots, w_q)$ are not known and are to be estimated by the learning system.

By assuming independence and identical distribution among the data in X, we can write the joint density function of X as the product of the densities of each data pair:

$$L(\mathbf{w}, X) = \prod_{i=1}^{N} \mu_i^{y_i}(1-\mu_i)^{1-y_i}, \tag{2.22}$$

where $\mu_i = f(\mathbf{w}^t\mathbf{x}_i)$. Equation (2.22) represents the *likelihood* function of X. Our objective is to maximize $L(\mathbf{w}, X)$ with respect to $\mathbf{w}$. By computing the log of (2.22) we obtain $\log L(\mathbf{w}, X) = l(\mathbf{w}, X) = \sum_{i=1}^{N}[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]$, and thus the cost function

$$J(\mathbf{w}) = -(\sum_{i=1}^{N}[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]), \tag{2.23}$$

which is the *binary cross entropy*. By taking the gradient with respect to the weights w_j we get

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{N}(y_i - \mu_i)x_{ij}, \tag{2.24}$$

where x_{ij} is the jth component of $\mathbf{x}_i$, and $f(z)$ is the *logistic function*:

$$f(z) \stackrel{\text{def}}{=} \frac{1}{1+e^{-z}}. \tag{2.25}$$

Finally, by considering the stocastic gradient, we obtain the following learning rule for the weights

$$\Delta w_j \propto (y - \mu)x_j. \tag{2.26}$$

The use of the logistic function as the output function for a two case classification problem has a motivation within Bayes theory: given specific assumptions, the logistic function formally provides the posterior probability of a given class [Jor95]. A similar result can be obtained for the general classification problem with more than two groups, where the logistic function is replaced by the *softmax function* and the cost function gets the general form of a cross entropy.

2.3.2 Decision Trees

Decision trees constitute a local learning paradigm that employs local averaging to estimate the class posterior probabilities for the decision rule (2.8) [BFO+84, Qui86, Qui93]. The regions over which the averaging takes place are constructed in a highly adaptive manner by using a top-down recursive splitting strategy, from which the alternative name of *recursive partitioning* (RP) is derived. RP begins with a single region R_0 containing all the training data. At each step every existing region is split into two subregions, thereby increasing the number of regions. This recursive splitting procedure is continued until a region meets a local terminal criterion, i.e. it contains train-

ing observations of the same class, and is not further split. When all regions have met the terminal criterion, they provide the final input space partition for local estimation of class probabilities.

The shape of the terminal regions is governed by the splitting procedure. One defines a splitting function $g(\mathbf{x}, \mathbf{a})$ of the input variables $\mathbf{x}$, characterized by a set of parameters $\mathbf{a}$, and a real valued split point s. The form of the split function is usually taken to be linear

$$g(\mathbf{x}, \mathbf{a}) = \mathbf{a}^t \mathbf{x} \tag{2.27}$$

(CART) [BFO+84], often with the restriction of being parallel to the coordinate input axes (CART, C4.5) [BFO+84, Qui93]

$$\mathbf{a} \in \{\mathbf{e}_1, \cdots, \mathbf{e}_n\}. \tag{2.28}$$

Here $\mathbf{e}_j$ is the basis unit vector for the jth input coordinate. The particular variable j and location s used to define the split are those that jointly minimize a given criterion. In general, such criterion is related to the average misclassification risk associated with using the resulting subregions to classify all training observations that respectively lie within them. In other words, the variable j and location s that best separate the data in the current region are chosen.

After the splitting is completed a "pruning" procedure is usually applied that recursively recombines adjacent regions in a bottom-up manner using cross-validated misclassification risk to determine when to stop the pruning. Other approaches are also possible. For example, all available data can be used for training, and a statistical test (e.g., chi-square) is then applied to estimate whether pruning a particular node is likely to produce an improvement beyond the training set. Techniques that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data, are also in use [Mit97].

The resulting set of regions represents a disjoint partition of the input measurement space. The region used for estimating the class probabilities and making the assignment (2.8) for any prediction point $\mathbf{x}_0$ is the resulting region in which the point lies.

We note that at each step in the recursive subdivision the variable that gives the most estimated classification information is considered for splitting, thereby reducing the extent of the neighborhood along the chosen axis for all successive regions that are descendants of the one being split. Moreover, the choice of the variable for splitting is based only on the local data contained in each region. Thus, recursive partitioning methods can exploit "local relevance" of input variables at different query points $\mathbf{x}_0$ in feature space. Some recursive partitioning implementations add locally derived variables that are functions of the original input variables, allowing regions to have non-axis oriented boundaries. Finally, if pruning is employed recursive partitioning methods also estimates the best region size locally for different query points $\mathbf{x}_0$.

With this built-in flexibility one might expect recursive partitioning methods to perform well in general. However, in benchmark studies even the simplest K-NN rules (see Section 2.3.3) often outperform recursive partitioning methods. Among the possible reasons for this behavior, some can be identified. The training sample size N restricts the number of splits that can be used to limit the extension of a subregion. As the partitioning proceeds the number of training observations in successive regions becomes smaller and smaller. After a relatively small number of splits the data remaining is insufficient to provide meaningful estimates. A second limitation, related to the first, is that decision trees produce a *partition* of the input space, that is the terminal regions that cover the space are disjoint. The consequence is that the resulting approximations are piecewise-constant and discontinuous, leading to large errors (bias) near region boundaries. Each prediction point is

contained in only one region and it can be very far from the region center and training points in that region. This can induce high bias that may overcome the bias reduction achieved by customizing the shape of the region. Finally, the approximations produced by RP are highly unstable with respect to minor perturbations of the training data. This leads to high variance predictions, induced by sampling fluctuations of the mechanism that produces the training data. Minor changes in an early split can have a major impact on later splits producing very different terminal regions.

Nearest neighbor methods, to be introduced next, overcome some of the limitations of recursive partitioning techniques by producing continous and overlapping neighborhoods, resulting in highly stable procedures with respect to perturbations of the training data. It should be also mentioned that several approaches have been proposed to mitigate the instability of RP methods. Among them is the *bagging* procedure [Bre96, Qui96].

2.3.3 Nearest-Neighbor Methods

The K nearest neighbor classification method [CD88, Ho98, Low95, Mcl92, Sal91, Sto77] is a simple and appealing approach: it finds the K nearest neighbors of the query point $\mathbf{x}_0$ in the training set, and then predicts the class label of $\mathbf{x}_0$ as the most frequent one occurring in the K neighbors. Such a method produces continuous and overlapping neighborhoods, and uses a different neighborhood for each individual query so that all points in the neighborhood are close to the query, to the extent possible. It is based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities for a classification problem. That is, $f_j(\mathbf{x} + \delta\mathbf{x}) \simeq f_j(\mathbf{x})$ for $||\delta\mathbf{x}||$ small enough, where $\{f_j(\mathbf{x})\}_{j=1}^J = \{P(j|\mathbf{x})\}_{j=1}^J$. Then,

$$f_j(\mathbf{x}_0) \simeq \frac{1}{|N(\mathbf{x}_0)|} \sum_{\mathbf{x} \in N(\mathbf{x}_0)} f_j(\mathbf{x}) \tag{2.29}$$

where $N(\mathbf{x}_0)$ is a neighborhood of $\mathbf{x}_0$ that contains points $\mathbf{x}$ in the q dimensional space that are "close" to $\mathbf{x}_0$. $|N(\mathbf{x}_0)|$ denotes the number of points in $N(\mathbf{x}_0)$. Given the training data $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, this motivates the estimates

$$\hat{f}(j|\mathbf{x}_0) = \frac{\sum_{n=1}^N 1(\mathbf{x}_n \in N(\mathbf{x}_0))1(y_n = j)}{\sum_{n=1}^N 1(\mathbf{x}_n \in N(\mathbf{x}_0))}, \tag{2.30}$$

where $1(\cdot)$ is an indicator function such that it returns 1 when its argument is true, and 0 otherwise.

A particular nearest-neighbor method is defined by how the neighborhood $N(\mathbf{x}_0)$ is specified. K-nearest-neighbor methods (K-NN) define the region at $\mathbf{x}_0$ to be the one that contains exactly the K closest training points to $\mathbf{x}_0$ according to a p-norm distance metric on the Euclidean space of the input measurement variables

$$D_p(\mathbf{x}_0, \mathbf{x}) = \{\sum_{i=1}^q |[W(\mathbf{x}_0)(\mathbf{x}_0 - \mathbf{x})]_i|^p\}^{1/p}. \tag{2.31}$$

The resulting neighborhood is determined by the value of K and by the choice of the distance measure, which in turn depends on a norm $p > 0$ and a metric defined by the matrix $W(\mathbf{x}_0) \in \Re^{q \times q}$.

The K-nearest-neighbor method has nice asymptotic properties [CH67, Cov68]. As the training sample size N becomes arbitrarily large it results

$$\lim_{N\to\infty} \hat{f}_j(\mathbf{x}) = f_j(\mathbf{x}) \tag{2.32}$$

provided that the value for K is chosen as a function of N so that

$$\lim_{N\to\infty} K = \infty, \qquad \lim_{N\to\infty} K/N = 0. \tag{2.33}$$

The first condition reduces the variance by making the estimation independent of the accidental characteristics of the K nearest neighbors. The second condition reduces the bias by assuring that

the K nearest neighbors are arbitrarily close to the query point. Intuitively, if K is kept fixed when the number N of samples is allowed to approach infinity, then all the K nearest neighbors will converge to $\mathbf{x}$. Conditions (2.33) require that K grow infinitely large, but at a lower rate than N.

These results imply that asymptotically, as $N \to \infty$, the K-NN rule achieves the minimum error rate of the Bayes rule, provided that the value of K is properly chosen, independent of the norm p or the metric $W(\mathbf{x})$.

Another asymptotic result holds for the 1-NN rule. Let E_1 be the classification error rate of the 1-NN rule and E_∞ that of the Bayes rule. Then $\lim_{N\to\infty} E_1 = e$, where $E_\infty \leq e \leq E_\infty(2 - e) \leq 2E_\infty$, so that with an unlimited number of samples the error rate for the 1-NN rule is no worse than twice the Bayes rate. This implies that in the asymptotic limit no decision rule is more than twice as accurate as the 1-NN rule, including a K-NN rule for any value of K.

Let us gain some understanding on why the 1-NN rule should work well. Let $\mathbf{x}_0$ be the query point and $\mathbf{x}$ its nearest neighbor in the training data. We recall that the label y of $\mathbf{x}$ is a random variable, and the probability that $y = j$ is simply the posterior probability $P(j|\mathbf{x})$. Then, when the number of samples is very large, it is reasonable to assume that $\mathbf{x}$ is sufficiently close to $\mathbf{x}_0$ so that $P(j|\mathbf{x}) \simeq P(j|\mathbf{x}_0)$. In this prospective we can view the 1-NN rule as a randomized decision that classifies $\mathbf{x}_0$ by selecting the class j with probability $P(j|\mathbf{x}_0)$.

If we define $j^*(\mathbf{x})$ as

$$P(j^*|\mathbf{x}) = \max_j P(j|\mathbf{x})$$

then the Bayes rule always selects j^*. When $P(j^*|\mathbf{x})$ is close to one, the nearest-neighbor selection is almost always the same as the Bayes selection. That is, when the minimum probability of error

is small, the nearest-neighbor probability of error is also small. When $P(j^*|\mathbf{x})$ is close to $1/J$, so that all classes are essentially equally likely, the selections made by the nearest-neighor rule and the Bayes rule are rarely the same, but the probability of error is approximately $1 - 1/J$ for both.

Example. Consider two classes (labelled 1 and 2) with probability density functions f_1 and f_2, respectively, as in Figure 2.1. Each density function is a mixture of two uniform distributions. Both densities share the same component for $x \in [3, 5]$; in this interval both densities have value $p < 1/2$. Assume that the prior probabilities of the two classes are equal to 1/2. The Bayes risk at x, when $x \in [3, 5]$, is

$$r^*(x) = \sum_{j=1}^{2} P(j|x)(1 - P(j|x)) = 1/2$$

Since $x \in [3, 5]$ with probability $2p$, the Bayes risk is equal to p. If $x \in [1, 2]$, it definetely belongs to class 1. With probability 1, the training set contains at least one sample of class 1 in the interval $[1, 2]$. This is because the training set size is growing to infinity. Hence, the 1-NN rule will classify x correctly. A similar argument applies when $x \in [6, 7]$. However, when $x \in [3, 5]$, $P(1|x) = 1/2$, that is the conditional probability that the class label is 1 given x is 1/2, because the priors are equals, and the class conditional densities are equal in such interval. Therefore, no matter how the 1-NN classifies a sample $x \in [3, 5]$, it is wrong with probability 1/2, just like the Bayes decision rule. Hence, the 1-NN rule has the same conditional probability of error as the Bayes decision rule for every x, and therefore its risk equals the Bayes risk.

The asymptotic results discussed suggest that the 1-NN based on simple Euclidean distance ($p = 2, W(\mathbf{x}_0) = I$)

$$D_2(\mathbf{x}_0, \mathbf{x}) = [\sum_{i=1}^{q} (x_{0i} - x_i)^2]^{1/2} \tag{2.34}$$

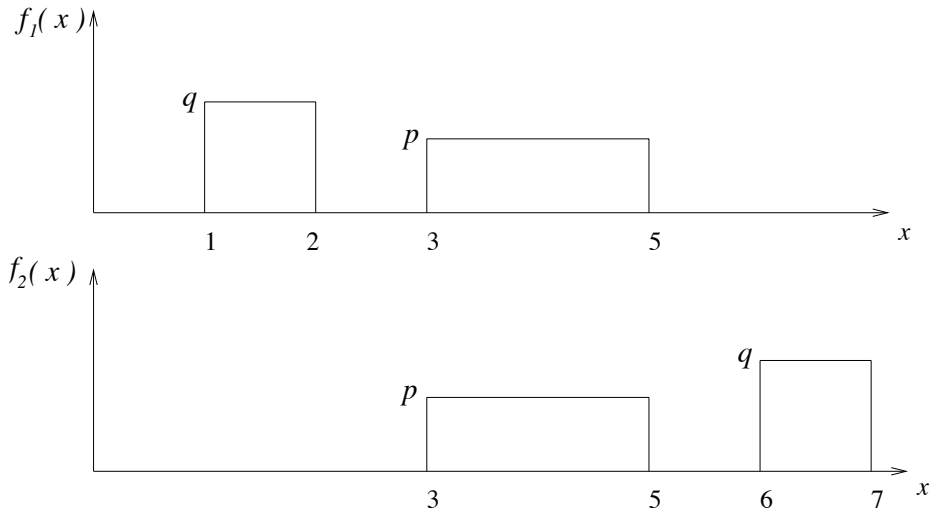

Figure 2.1: Probability density functions $f_1(x)$ and $f_2(x)$.

might perform well provided that the training data set is not too small.

An obvious question concerns the rate of convergence of the 1-NN procedure: how rapidly the performance of the 1-NN rule converges to the asymptotic value? Unfortunately, the only statements that can be made in the general case are negative. It can be shown that convergence can be arbitrarily slow, and the error rate need not even decrease monotonically with the increasing number of data [Cov68]. As with other nonparametric methods, it is difficult to obtain anything other than asymptotic results without making regularity assumptions about the underlying probability structure. Under suitable smoothness conditions, the error rate of the N sample nearest neighbor rule converges to its limit on the order of $1/N^2$: $E_1(N) = E_\infty + O(1/N^2)$ [Cov68].

2.4 Support Vector Machines

In this section we introduce the main concepts and properties of *support vector machines* (SVMs) [Bur98, Vap98, Vap99, CT01, SS02]. Again, we are given N observations. Each observa-

tion consists of a pair: a vector $\mathbf{x}_i \in \Re^q, i = 1, \ldots, N$, and the associated class label $y_i \in \{-1, 1\}$ (we restrict ourselves to a two class case). It is assumed that there exists some unknown probability distribution $P(\mathbf{x}, y)$ from which these data are drawn. The task it to learn the set of parameters α in $f(\mathbf{x}, \alpha)$ so that f realizes the mapping $\mathbf{x}_i \rightarrow y_i$. A particular choice of α defines the corresponding trained machine $f(\mathbf{x}, \alpha)$.

The expectation of the test error (i.e., the expected risk, or just the risk) for a trained machine is

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{x}, \alpha)|dP(\mathbf{x}, y).$$

This quantity gives a nice way of writing the true mean error, but unless we have an estimate of what $P(\mathbf{x}, y)$ is, it is not very useful. The empirical risk $R_{emp}(\alpha)$ is then defined, as the mean error rate measured over the training set:

$$R_{emp}(\alpha) = \frac{1}{2N} \sum_{i=1}^{N} |y_i - f(\mathbf{x}_i, \alpha)|.$$

The following bound holds (with high probability over the random draw of the training sample) [Vap99]:

$$R(\alpha) \leq R_{emp}(\alpha) + Conf(h),$$

where h is the Vapnik Chervonenkis (VC) dimension, and is a measure of the ability of the machine to learn any training set without error. The term $Conf(h)$ is called the VC confidence. Given a family of functions $f(\mathbf{x}, \alpha)$, it is desirable to choose the machine which gives the lowest upper bound on the risk. The first term, $R_{emp}(\alpha)$, represents the accuracy attained on a particular training set, whereas the second term $Conf(h)$ represents the ability of the machine to learn any training set without error. $R_{emp}(\alpha)$ and $Conf(h)$ respectively drive the bias and the variance of the generaliza-

tion error. The best generalization error is achieved when the right balance between these two terms is attained. This gives a principled method for choosing a learning machine for a specific task, and is the essential idea of structural risk minimization.

Unlike traditional methods which minimize the empirical risk, a support vector machine aims at minimizing the above upper bound of the generalization error. It achieves this goal by learning the αs in $f(\mathbf{x}, \alpha)$ so that the resulting trained machine satifies the maximum margin property, i.e. the decision boundary it represents has the maximum minimum distance from the closest training point.

The well developed theory that has motivated SVMs makes them an attractive learning machine for a variety of tasks. An SVM maps the data into a higher-dimensional space (feature space) and defines a separating hyperplane there. Translating the training set into a higher-dimensional space incurs both computational and learning-theoretic costs. SVMs avoid overfitting by choosing a particular hyperplane among the many that can separate the data in the feature space, specifically the maximum margin hyperplane.

The computational burden of explicitly representing the feature vectors is avoided by defining a function, called the *kernel function*, that plays the role of the dot product in feature space. Therefore, an SVM can locate a separating hyperplane in feature space and classify points in that space without ever representing the space explicitly.

By choosing different functions as kernels, SVMs can realize Radial Basis Function (RBF), Polynomial and Multi-layer Perceptron classifiers. Compared with the traditional way of implementing such classifiers, SVMs have the advantage of automatically selecting both the num-

ber and locations of the kernel function during training.

In addition to avoid overfitting, the use of the maximum margin hyperplane leads to a learning algorithm that can be reduced to a convex optimization problem. In order to train the SVM, the unique minimum of a convex function must be found. As a consequence, support vector machines do not suffer from the local minima problem that affects many learning schemes and, unlike the backpropagation learning algorithm for neural networks, a given SVM will always deterministically converge to the same solution for a given data set, regardless of the initial conditions.

Another appealing feature of SVMs is the sparseness representation of the decision boundary they provide. The location of the separating hyperplane in feature space is specified via real-valued weights on the training examples. In general, those training examples that lie far away from the hyperplane do not participate in its specification and therefore receive zero weight. Training examples that lie close to the decision boundary between the two classes receive non-zero weights. These training examples are called *support vectors*, since their removal would change the location of the separating hyperplane. The design of SVMs, in general, allows the number of support vectors to be small compared to the total number of training examples. This property allows the SVM to classify new examples efficiently, since the majority of the training examples will be safely ignored.

2.4.1 Learning with SVMs

In the simple case of two linearly separable classes, a support vector machine selects, among the infinite number of linear classifiers that separate the data, the classifier that minimizes an upper bound on the generalization error. The SVM achieves this goal by computing the classifier that satifies the maximum margin property, i.e. the classifier whose decision boundary has the

maximum minimum distance from the closest training point.

If the two classes are non-separable, the SVM looks for the hyperplane that maximizes the margin and that, at the same time, minimizes a quantity proportional to the number of misclassification errors. The trade-off between margin and misclassification error is driven by a positive constant C that has to be chosen beforehand. The corresponding decision function is then obtained by considering the $sign(f(\mathbf{x}))$, where $f(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i^T \cdot \mathbf{x} - b$, and the coefficients α_i are the solution of a convex quadratic problem, defined over the hypercube $[0, C]^N$. In general, the solution will have a number of coefficients α_i equal to zero, and since there is a coefficient α_i associated to each data point, only the data points corresponding to non-zero α_i will influence the solution. These points are the support vectors. Intuitively, the support vectors are the data points that lie at the border between the two classes, and a small number of support vectors indicates that the two classes can be well separated.

This technique can be extended to allow for non-linear decision surfaces. This is done by mapping the input vectors into a higher dimensional feature space: $\phi : \Re^q \rightarrow \Re^Q$, and by formulating the linear classification problem in the feature space. Therefore, $f(\mathbf{x})$ can be expressed as $f(\mathbf{x}) = \sum_i \alpha_i y_i \phi^T(\mathbf{x}_i) \cdot \phi(\mathbf{x}) - b$.

If one were given a function $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, one could learn and use the maximum margin hyperplane in feature space without having to compute explicitly the image of points in $\Re^Q$. It has been proved (Mercer's Theorem) that for each continuous positive definite function $K(\mathbf{x}, \mathbf{y})$ there exists a mapping ϕ such that $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y}), \forall \mathbf{x}, \mathbf{y} \in \Re^q$. By making use

of such function K (kernel function), the equation for $f(\mathbf{x})$ can be rewritten as

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) - b. \tag{2.35}$$

Examples of kernel functions are:

$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$ (polynomial);

$K(\mathbf{x}, \mathbf{y}) = e^{-||\mathbf{x}-\mathbf{y}||^2/2\sigma^2}$ (Gaussian);

$K(\mathbf{x}, \mathbf{y}) = tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta)$ (sigmoid).

Chapter 3

Related Work

Pattern classification faces a difficult challenge in finite settings and high dimensional spaces due to the curse of dimensionality. It becomes crucial in such cases the estimation of different degrees of relevance that input features may have in various locations in feature space. In this chapter we discuss previous work in the literature on flexible metric computation.

3.1 Curse-of-dimensionality

Related to the question of rate of convergence of the 1-NN rule is the one on how well the rule works in finite-sample settings. The asymptotic results rely on the fact that the bias of the estimate of each $f_j(\mathbf{x})$

$$bias\hat{f}_j(\mathbf{x}) = f_j(\mathbf{x}) - E[\hat{f}_j(\mathbf{x})] \tag{3.1}$$

becomes arbitrarily small. This is because the region $N(\mathbf{x}_0)$ will only contain training points $\mathbf{x}$ arbitrarily close to $\mathbf{x}_0$ (provided that $f_j(\mathbf{x})$ is continuous at $\mathbf{x}_0$ and $K/N \rightarrow 0$). In a finite setting, if

the number of training data N is large and the number of input features q is small then the asymptotic results may still be valid. However, for a moderate to large number of input variables, the sample size required for their validity is usually beyond feasibility.

This phenomenon is known as the *curse-of-dimensionality* [Bel61]. It refers to the fact that in high dimensional spaces data become extremely sparse and are far apart from each other. To get a quantitive idea of this phenomenon, consider a random sample of size N drawn from a uniform distribution in the q dimensional unit hypercube. The expected diameter of a $K = 1$ neighborhood using Euclidean distance is proportional to $N^{-1/q}$, which means that for a given q, the diameter of the neighborhood containing the closest training point shrinks as $N^{-1/q}$ for increasing N. Table 3.1 shows the length d of the diameter for various values of q and N. For example, for $q = 20$, if $N = 10^4$ the length d of the diameter is 1.51, if $N = 10^6$ $d = 1.20$, if $N = 10^{10}$ $d = 0.76$. Considering that the entire range of each variable is 1, we note that even for a moderate number of input variables very large training sample sizes are required to make a $K = 1$ nearest neighborhood relatively small. The proportion (diameter $\sim N^{-1/q}$) discussed here for $p = 2$ inflicts all $p > 0$ norms.

The fact that the data become so sparse in high dimensional spaces has the consequence that the bias of the estimate can be quite large even for $K = 1$ and very large data sets. This high bias effect due to the curse-of-dimensionality can be reduced by taking into consideration the fact that the class probability functions may not vary with equal strength in all directions in the feature space emanating from the query point $\mathbf{x}_0$. This can be accomplished by choosing a metric $W(\mathbf{x}_0)$ (2.31) that credits the highest influence to those directions along which the class probability functions are not locally constant, and correspondingly less influence to other directions. As a result, the class

Table 3.1: Expected length of the diameter d of a $K = 1$ neighborhood for various values of q and N.

q	N	$d(q, N)$
4	100	0.42
4	1000	0.23
6	100	0.71
6	1000	0.48
10	1000	0.91
10	10^4	0.72
20	10^4	1.51
20	10^6	1.20
20	10^{10}	0.76

conditional probabilities tend to be approximately constant in the resulting modified neighborhood, whereby better classification can be obtained, as we will see later.

From the above discussion it should be clear that, in finite settings, the choice of the metric $W(\mathbf{x}_0)$ can strongly affect performance, and therefore the choice of a distance measure becomes crucial in determining the outcome of nearest neighbor classification.

Nevertheless, it should be noted that, despite the curse-of-dimensionality, nearest neighbor methods in many benchmark studies turn out to be competitive with other classification methods, and often are among the best performers. There may be many reasons for this behavior but two can be clearly identified. First, often there is a high degree of correlation among the input features so that the data actually lie within a (much) lower dimensional subspace of the q-dimensional measurement space. It is this intrinsic dimensionality of the data that drives the curse-of-dimensionality: to the extent that it is much smaller than the number of inputs the curse is mitigated. A second reason is due to the nature of the classification problem itself. For accurate classification (2.8) it is not

necessary to achieve accurate estimates of the actual values of the conditional probabilities. For example, with the zero-one loss function (2.3) to obtain the optimal decision (2.5) is sufficient that the largest estimated class probability correspond to the one that is actually the largest

$$\max_{1\leq j\leq J} \hat{f}_j(\mathbf{x}) = \max_{1\leq j\leq J} f_j(\mathbf{x}) \tag{3.2}$$

irrespective of their actual values, or the values and order of the estimated probabilities for the other classes. Therefore, the bias (3.1) could be very high but if it affects all the estimates $\hat{f}_j(\mathbf{x})$ in roughly the same proportion, so that their order relation is similar enough to that of the true underlying probabilities $f_j(\mathbf{x})$, an optimal class assignment can still be performed.

3.2 Adaptive Metric Techniques

While K-NN methods are more resistant to the curse-of-dimensionality than expected, their performance is still affected by it. The need for large neighborhoods in high dimensional spaces can affect the bias (3.1) differentially for the respective class probability estimates enough to cause nonoptimal decision, thereby increasing misclassification error. Thus, the nearest neighbor rule becomes less appealing with finite training samples. Severe bias can be introduced in the nearest neighbor rule in a high dimensional input feature space with finite samples. The commonly used Euclidean distance measure, while simple computationally, implies that the input space is isotropic or homogeneous. However, the assumption for isotropy is often invalid and generally undesirable in many practical applications. Figure 3.1 illustrates a case in point, where class boundaries are parallel to the coordinate axes. For query a, dimension X is more relevant, because a slight move along the X axis may change the class label, while for query b, dimension Y is more relevant. For

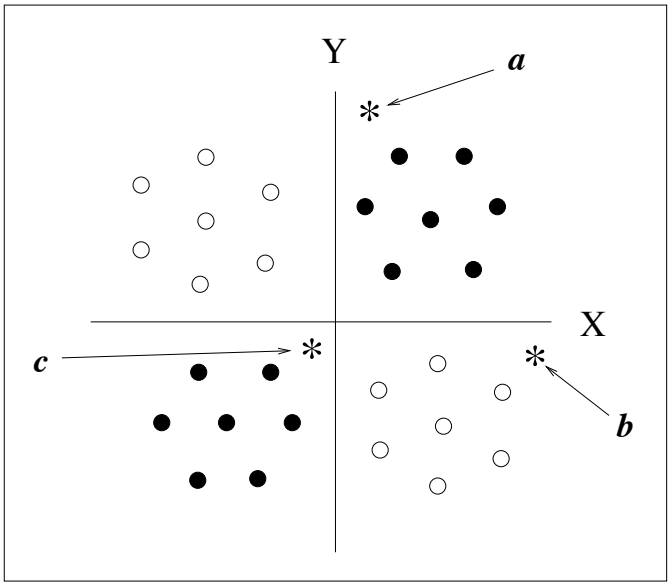

Figure 3.1: Feature relevance varies with query locations.

query c, however, both dimensions are equally relevant. This implies that distance computation does not vary with equal strength or in the same proportion in all directions in the feature space emanating from the input query. Capturing such information, therefore, is of great importance to any classification procedure in high dimensional settings.

3.3 Flexible Metric Nearest Neighbor Classification

Friedman [Fri94] describes an adaptive approach for pattern classification that combines some of the best features of K-NN learning and recursive partitioning. The resulting hybrid method inherits the flexibility of recursive partitioning to adapt the shape of a region $N(\mathbf{x}_0)$ as well as the ability of nearest neighbor techniques to keep the points within the region close to the point being predicted. The method is capable of producing nearly continuous probability estimates with the

region $N(\mathbf{x}_0)$ centered at $\mathbf{x}_0$, and the shape of the region separately customized for each individual prediction point. In the following we describe the method proposed in [Fri94] in more details.

Consider an arbitrary function $f(\mathbf{x})$ of q arguments $(x_1, \cdots, x_q)$. In the absence of values for any of the argument variables the least-squares estimate for $f(\mathbf{x})$ is just the expected value $Ef = \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x}$, over the joint probability density of its arguments. Suppose now that the value of just one of the argument variables x_i were known, say $x_i = z$. The least-squares prediction for $f(\mathbf{x})$ in this case would be the expected value of $f(\mathbf{x})$, under the restriction that x_i assumes the known value z: $E[f|x_i = z] = \int f(\mathbf{x})p(\mathbf{x}|x_i = z)d\mathbf{x}$. The improvement in squared prediction error $I_i^2(z)$ associated with knowing the value z of the ith input variable $x_i = z$ is therefore

$$I_i^2(z) = (Ef - E[f|x_i = z])^2. \tag{3.3}$$

$I_i^2(z)$ measures how much we gain by knowing that $x_i = z$. It reflects the influence of the ith input variable on the variation of $f(\mathbf{x})$ at the particular point $x_i = z$. Note that if $Ef = E[f|x_i = z]$ then $f(\mathbf{x})$ is independent of x_i at the particular point $x_i = z$, and accordingly $I^2(z) = 0$.

Consider an arbitrary point $\mathbf{z} = (z_1, \cdots, z_q)$ in the q-dimensional input space. A measure of the relative influence, *relevance*, of the ith input variable x_i to the variation of $f(\mathbf{x})$ at $\mathbf{x} = \mathbf{z}$ is given by

$$r_i^2(\mathbf{z}) = \frac{I_i^2(z_i)}{\sum_{k=1}^{q} I_k^2(z_k)}. \tag{3.4}$$

In [Fri94], Friedman proposes an algorithm, called *machete*, that uses the local relevance measure (3.4) to define a splitting procedure centered at the prediction point, overcoming some of the limitations of the static splitting of recursive partitioning. As with recursive partitioning, the machete begins with the entire input measurement space R_0 and divides it into two regions by a

split on one of the input variables. However, the manner in which the splitting variable is selected and the nature of the split itself, are quite different. The input variable used for splitting is the one that maximizes the estimated relevance as evaluated at the point $\mathbf{z}$ to be predicted

$$i^*(\mathbf{z}) = \arg \max_{1 \leq i \leq q} \hat{r}_i^2(\mathbf{z}). \tag{3.5}$$

Thus, for the same training data, different input variables can be selected for this first split at different prediction points $\mathbf{z}$, depending on how the relevance of each input variable changes with location in feature space. The space is then split on the i^*th input variable so that the i^*th component of $\mathbf{z}$, z_{i^*}, is centered within the resulting subinterval that contains it. In particular the training data are sorted in increasing order on $|z_{i^*} - x_{ni^*}|$ and the new region $R_1(\mathbf{z})$ is

$$R_1(\mathbf{z}) = \{\mathbf{x}_n \mid |z_{i^*} - x_{ni^*}| \leq d(M_1)\} \tag{3.6}$$

where $d(M_1)$ is a distance value such that $R_1(\mathbf{z})$ contains $M_1 < N$ training observations.

As with all recursive methods, the entire machete procedure is defined by successively applying its splitting procedure to the result of the previous split. The algorithm stops when there are K training observations left in the region under consideration, with K being one of the input parameters of the machete.

In [Fri94], Friedman also proposes a generalization of the machete algorithm, called *scythe*, in which the input variables influence each split in proportion to their estimated local relevance, rather than according to the winner-take-all strategy of the machete.

The major limitation concerning the machete/scythe method is that, like recursive partitioning methods, it applies a "greedy" strategy. Since each split is conditioned on its "ancestor"

split, minor changes in an early split, due to any variability in parameter estimates, can have a significant impact on later splits, thereby producing different terminal regions. This makes the predictions highly sensitive to the sampling fluctuations associated with the random nature of the process that produces the training data, and therefore may lead to high variance predictions.

We performed a comparative study, to be seen later (Chapter 5), that shows that while machete/scythe demonstrates performance improvement over recursive partitioning, simple K-NN still remains highly competitive.

3.4 Discriminant Adaptive Nearest Neighbor Classification

In [HT96a], Hastie and Tibshirani propose a discriminant adaptive nearest neighbor classification method (DANN) based on linear discriminant analysis. The method computes a local distance metric as a product of properly weighted within and between sum of squares matrices. The authors also describe a method to perform global dimensionality reduction, by pooling the local dimension information over all points in the training set [HT96a, HT96b].

The goal of linear discriminant analysis (LDA) is to find an orientation in feature space on which the projected training data are well separated. This is obtained by maximizing the difference between the class means relative to some measure of the standard deviations for each class. The difference between the class means is estimated by the *between-class scatter matrix* B, and the measure of the standard deviations for each class is given by the *within-class scatter matrix* W. Both matrices are computed by using the given training data. Once the data are rotated and scaled for best separation of classes, a query point is classified to the class of the closest centroid, with a correction for the class prior probabilities.

In [HT96a], the authors estimate B and W locally at the query point, and use them to form a metric that behaves locally like the LDA metric. The metric proposed is $\Sigma = W^{-1}BW^{-1}$, which has the effect of crediting larger weights to directions in which the centroids are more spread out than to those in which they are close. First the metric Σ is initialized to the identity matrix. A nearest neighborhood of K_m points around the query point $\mathbf{x}_0$ is identified using the metric Σ. Then, the weighted within and between sum of squares matrices W and B are calculated using the points in the neighborhood of $\mathbf{x}_0$. The result is a new metric $\Sigma = W^{-1}BW^{-1}$ for use in a nearest neighbor classification rule at $\mathbf{x}_0$. The algorithm can be either a single step procedure, or a larger number of iterations can be carried on.

The authors also show that the resulting metric used in DANN approximates the weighted *Chi-squared* distance

$$D(\mathbf{x}, \mathbf{x}_0) = \sum_{j=1}^{J} \frac{[P(j|\mathbf{x}) - P(j|\mathbf{x}_0)]^2}{P(j|\mathbf{x}_0)}, \tag{3.7}$$

which measures the distance between the query point $\mathbf{x}_0$ and its nearest neighbor $\mathbf{x}$, in terms of their class posterior probabilities. The approximation, derived by a Taylor series expansion, holds only under the assumption of Gaussian class densities with equal covariance matrices.

While sound in theory, DANN may be limited in practice. The main concern is that in high dimensions we may never have sufficient data to fill in $q \times q$ matrices. Also, the fact that the distance metric computed by DANN approximates the weighted *Chi-squared* distance (3.7) only when class densities are Gaussian and have the same covariance matrix may cause a performance degradation in situations where data do not follow Gaussian distributions or are corrupted by noise, which is often the case in practice. We will see that this hypothesis is validated in our experimental results (Chapter 5). These considerations lead us to our method.

Chapter 4

Adaptive Metric Nearest Neighbor Classification

This chapter presents a novel approach (ADAMENN) to computing local feature relevance. The technique uses the *Chi-squared* distance in order to estimate to which extent each dimension can be relied on to predict class posterior probabilities. We motivate our technique and describe the algorithm in detail.

4.1 Introduction

K-NN methods are based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities for a classification problem. This assumption, however, becomes invalid for any fixed distance metric when the input observation $\mathbf{x}_0$ is near a class boundary, as illustrated in Figure 3.1. In the following, we describe a nearest

neighbor classification technique that is capable of producing a local neighborhood in which the posterior probabilities are approximately constant, and that is highly adaptive to query locations [DPG00a, DPG00b, DPG02].

We propose an adaptive nearest neighbor classification method to try to minimize bias in high dimensions. We estimate a flexible metric for computing neighborhoods based on *Chi-squared* distance analysis. The resulting neighborhoods are highly adaptive to query locations. Moreover, the neighborhoods are elongated along less relevant feature dimensions and constricted along most influential ones. As a result, the class conditional probabilities tend to be constant in the modified neighborhoods, whereby better classification performance can be obtained.

Figure 4.1 shows an example. There are two classes and the data for both classes are generated from a bivariate standard normal distribution. The data for class one have the radius less than or equal to 1.15, while the data for class two have the radius greater than 1.15. As a result, class one is surrounded by class two. Figure 4.1(a) shows the nearest neighborhood of size 50 of a query located at (0, -1) near the class boundary. This neighborhood is computed using the Euclidean distance metric. Figure 4.1(b) shows the same size neighborhood computed by using our adaptive nearest neighbor classification algorithm. Note how the modified neighborhood is elongated along the direction of the true decision boundary and constricted along the direction orthogonal to it, which is the most relevant direction for the given query.

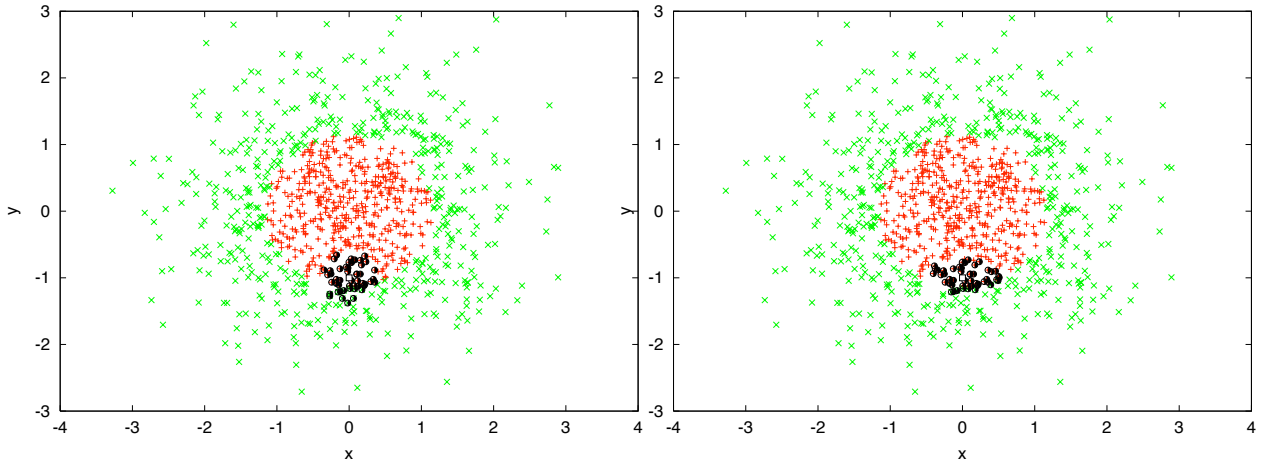

Figure 4.1: Plot (a) shows the spherical neighborhood of the query point $(0, -1)$ containing 50 points (shown as darker circles). Plot (b) shows the corresponding neighborhood found by our adaptive nearest neighbor algorithm, also containing 50 points. After applying our adaptive procedure the neighborhood is constricted along the most relevant dimension and elongated along the less important one.

4.2 Chi-Squared Distance

Our technique is motivated as follows. Consider a query point with feature vector $\mathbf{x}_0$. Let $\mathbf{x}$ be the nearest neighbor of $\mathbf{x}_0$ computed according to a distance metric $D(\mathbf{x}, \mathbf{x}_0)$. Our goal is to find a metric $D(\mathbf{x}, \mathbf{x}_0)$ that minimizes $E[r(\mathbf{x}_0, \mathbf{x})]$, where $r(\mathbf{x}_0, \mathbf{x}) = \sum_{j=1}^{J} P(j|\mathbf{x}_0)(1 - P(j|\mathbf{x}))$. Here $P(j|\mathbf{x})$ is the class conditional probability at $\mathbf{x}$. That is, $r(\mathbf{x}_0, \mathbf{x})$ is the finite sample error risk given that the nearest neighbor to $\mathbf{x}_0$ by the chosen metric is $\mathbf{x}$ [DH73]. Equivalently, we can minimize

$$E(r^*(\mathbf{x}_0) - r(\mathbf{x}_0, \mathbf{x}))^2, \tag{4.1}$$

where $r^*(\mathbf{x}_0) = \sum_{j=1}^{J} P(j|\mathbf{x}_0)(1 - P(j|\mathbf{x}_0))$ is the theoretical infinite sample risk at $\mathbf{x}_0$. By substituting this expression and that for $r(\mathbf{x}_0, \mathbf{x})$ into (4.1), we obtain the following metric that

minimizes (4.1) [MH90]:

$$D(\mathbf{x}, \mathbf{x}_0) = (\sum_{j=1}^{J} P(j|\mathbf{x}_0)(P(j|\mathbf{x}) - P(j|\mathbf{x}_0)))^2 \tag{4.2}$$

The idea behind this metric is that if the value of $\mathbf{x}$ for which $D(\mathbf{x}, \mathbf{x}_0)$ is small is selected, then the expectation (4.1) will be minimized.

This metric is related to the theory of the two class case developed in [SF81]. However, a major concern with the above metric is that it has a cancellation effect when all classes are equally likely [MH90]. Suppose that for two classes i and k, $P(i|\mathbf{x}_0) = P(k|\mathbf{x}_0)$, and $(P(i|\mathbf{x}) - P(i|\mathbf{x}_0)) \approx -(P(k|\mathbf{x}) - P(k|\mathbf{x}_0))$. Then, in equation 4.2 the summands for classes i and k will tend to cancel each other out, even if the magnitudes of $(P(i|\mathbf{x}) - P(i|\mathbf{x}_0))$ and $(P(k|\mathbf{x}) - P(k|\mathbf{x}_0))$ are large.

We can overcome this limitation by considering the *Chi-squared* distance [HT96a] $D(\mathbf{x}, \mathbf{x}_0) = \sum_{j=1}^{J} [P(j|\mathbf{x}) - P(j|\mathbf{x}_0)]^2$, which measures the distance between the query $\mathbf{x}_0$ and the point $\mathbf{x}$, in terms of the difference between the class posterior probabilities at the two points. Furhermore, by multiplying it by $1/P(j|\mathbf{x}_0)$ we obtain the following weighted *Chi-squared* distance

$$D(\mathbf{x}, \mathbf{x}_0) = \sum_{j=1}^{J} \frac{[P(j|\mathbf{x}) - P(j|\mathbf{x}_0)]^2}{P(j|\mathbf{x}_0)}. \tag{4.3}$$

Note that in comparison to the *Chi-squared* distance, the weights, $1/P(j|\mathbf{x}_0)$, in (4.3) have the effect of increasing the distance of $\mathbf{x}_0$ to any point $\mathbf{x}$ whose most probable class is unlikely to include $\mathbf{x}_0$. That is, if $j^* = \arg\max_j P(j|\mathbf{x})$, we have $P(j^*|\mathbf{x}_0) \approx 0$, as a consequence, it becomes highly improbable for any such point to be a nearest neighbor candidate. In general, such a weighting benefits any nearest neighbor classifier whose distance metric approximates the *Chi-squared* distance [HT96a].

Equation (4.3) computes the distance between the true and estimated posteriors. Our goal is to estimate the relevance of feature i by computing its ability to predict the class posterior probabilities locally at the query point. We do so by considering the expectation of $P(j|\mathbf{x})$ conditioned at a location along feature dimension i. Then, the *Chi-squared* distance (4.3) tells us the extent to which dimension i can be relied on to predict $P(j|\mathbf{x})$. Thus, Equation (4.3) provides us with a foundation upon which to develop a theory of feature relevance in the context of pattern classification.

4.3 Local Feature Relevance

Based on the above discussion, the details of our local feature relevance measure are as follows. We first notice that $P(j|\mathbf{x})$ is a function of $\mathbf{x}$. Therefore, we can compute the conditional expectation of $P(j|\mathbf{x})$, denoted by $\overline{P}(j|x_i = z)$, given that x_i assumes value z, where x_i represents the ith component of $\mathbf{x}$. That is,

$$\begin{aligned} \overline{P}(j|x_i = z) &= E[P(j|\mathbf{x})|x_i = z] \\ &= \int P(j|\mathbf{x})p(\mathbf{x}|x_i = z)d\mathbf{x} \end{aligned} \tag{4.4}$$

Here $p(\mathbf{x}|x_i = z)$ is the conditional density of the other input variables defined as $p(\mathbf{x}|x_i = z) = p(\mathbf{x})\delta(x_i - z)/\int p(\mathbf{x})\delta(x_i - z)d\mathbf{x}$, where $\delta(x - z)$ is the Dirac delta function having the properties $\delta(x - z) = 0 \quad if \quad x \neq z$ and $\int_{-\infty}^{\infty}\delta(x - z)dx = 1$. Let

$$r_i(\mathbf{z}) = \sum_{j=1}^{J}\frac{[P(j|\mathbf{z}) - \overline{P}(j|x_i = z_i)]^2}{\overline{P}(j|x_i = z_i)}. \tag{4.5}$$

$r_i(\mathbf{z})$ represents the ability of feature i to predict the $P(j|\mathbf{z})$s at $x_i = z_i$. The closer $\overline{P}(j|x_i = z_i)$ is to $P(j|\mathbf{z})$, the more information feature i carries for predicting the class posterior probabilities locally at $\mathbf{z}$.

We can now define a measure of feature relevance for $\mathbf{x}_0$ as

$$\bar{r}_i(\mathbf{x}_0) = \frac{1}{K_0} \sum_{\mathbf{z} \in N(\mathbf{x}_0)} r_i(\mathbf{z}), \tag{4.6}$$

where $N(\mathbf{x}_0)$ denotes the neighborhood of $\mathbf{x}_0$ containing the K_0 nearest training points, according to a given metric. $\bar{r}_i$ measures how well on average the class posterior probabilities can be approximated along input feature i within a local neighborhood of $\mathbf{x}_0$. Small $\bar{r}_i$ implies that the class posterior probabilities will be well captured along dimension i in the vicinity of $\mathbf{x}_0$. Note that $\bar{r}_i(\mathbf{x}_0)$ is a function of both the test point $\mathbf{x}_0$ and the dimension i, thereby making $\bar{r}_i(\mathbf{x}_0)$ a local relevance measure.

To formulate the measure of feature relevance as a weighting scheme, we first define

$$R_i(\mathbf{x}_0) = \max_j \{\bar{r}_j(\mathbf{x}_0)\} - \bar{r}_i(\mathbf{x}_0),$$

i.e., the more relevant dimension i is, the larger R_i becomes. A weighting scheme can then be given by

$$w_i(\mathbf{x}_0) = (R_i(\mathbf{x}_0))^t / \sum_{l=1}^{q} (R_l(\mathbf{x}_0))^t, \tag{4.7}$$

where $t = 1, 2$, giving rise to linear and quadratic weightings, respectively. In this thesis we propose the following exponential weighting scheme

$$w_i(\mathbf{x}_0) = \exp(cR_i(\mathbf{x}_0)) / \sum_{l=1}^{q} \exp(cR_l(\mathbf{x}_0)) \tag{4.8}$$

where c is a parameter that can be chosen to maximize (minimize) the influence of $\bar{r}_i$ on w_i. When $c = 0$ we have $w_i = 1/q$, thereby ignoring any difference between the $\bar{r}_i$'s. On the other hand, when c is large a change in $\bar{r}_i$ will be exponentially reflected in w_i. In this case, w_i is said to follow the Boltzmann distribution. The exponential weighting is more sensitive to changes in local feature

relevance (4.6) and gives rise to better performance improvement. In fact, it is more stable because it prevents neighborhoods from extending infinitely in any direction, i.e., zero weight. This, however, can occur when either linear or quadratic weighting is used. Thus, (4.8) can be used to compute the weight associated with each feature, resulting in the weighted distance computation

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{q} w_i (x_i - y_i)^2}. \tag{4.9}$$

These weights $\mathbf{w}_i$ enable the neighborhood to elongate less important feature dimensions, and, at the same time, to constrict the most influential ones. Note that the technique is *query-based* because weightings depend on the query [Aha97, AMS97].

An intuitive explanation for (4.5) and, hence, (4.6), goes as follows. Suppose that the value of $r_i(\mathbf{z})$ is small, which implies a large weight along dimension i. Consequently, the neighborhood is shrunk along that direction. This, in turn, penalizes points along dimension i that are moving away from z_i. Now, $r_i(\mathbf{z})$ can be small only if the subspace spanned by the other input dimensions at $x_i = z_i$ likely contains samples similar to $\mathbf{z}$ in terms of the class conditional probabilities. Then, a large weight assigned to dimension i based on (4.8) says that moving away from the subspace, hence from the data similar to $\mathbf{z}$, is not a good thing to do. Similarly, a large value of $r_i(\mathbf{z})$, hence a small weight, indicates that in the vicinity of z_i along dimension i one is unlikely to find samples similar to $\mathbf{z}$. This corresponds to an elongation of the neighborhood along dimension i. Therefore, in this situation in order to better predict the query, one must look farther away from z_i.

So far we have considered estimating feature relevance along each individual dimension, one at a time. However, there are situations where feature relevance can only be captured by exam-

ining several feature variables simultaneously. That is, feature variables are not independent, and there is a degree of correlation among them. It should be clear that, in the absence of any other information, determining which feature subsets should be examined to estimate local relevance adds considerable complexity to feature relevance computation. One way to decorrelate association among the features is to rotate the feature dimensions so that they coincide with the eigenvectors of a sample covariance matrix, as in [HT96a]. We do not perform such transformation in our experiments.

Using full weight matrices relates to transforming the sample space. Principal component analysis (PCA) [Mei72, DH73, Fuk90] (known in the communication theory literature as the Karhunen-Löeve expansion) can transform a sample space, re-representing it with a list of polynomial equations of the original features, ordered by nonincreasing variance. However, PCA's bias is not always appropriate for classification; features with low variance might actually have high predictive relevance. In fact, for the purposes of pattern classification, the most serious problem of the approaches to global dimensionality reduction (e.g., PCA) is that they are overly concerned with *faithful representation* of the data. Greatest emphasis is usually placed on those features or groups of features that have the greatest variability. But, for classification, we are interested in *discrimination*, not representation. Roughly speaking, the most interesting features are the ones for which the difference in the class means is large relative to the standard deviations, not the ones for which the standard deviations are large.

In addition, traditional feature selection algorithms select certain dimensions in advance. This can lead to a loss of information. As shown in Figure 3.1, a single feature may have different degrees of relevance from location to location in input space. As a consequence, each dimension

could be relevant to at least one of the classes. Thus, it may not always be feasible to prune off too many dimensions without incurring a loss of crucial information. We solve this problem by crediting to features the query-based weights defined in (4.8).

We observe that the reduction in prediction error considered by Friedman [Fri94], in our notation, can be described by

$$I_i^2(\mathbf{z}) = \sum_{j=1}^{J} (\overline{P}(j) - \overline{P}(j|x_i = z_i)])^2, \tag{4.10}$$

where $\overline{P}(j)$ represents the expected value of $P(j|\mathbf{x})$. This measure reflects the influence of the ith input variable on the variation of $P(j|\mathbf{x})$ at the particular point $x_i = z$. In this case, the most informative input variable is the one that gives the largest deviation from the average value of $P(j|\mathbf{x})$.

One of the key differences between our relevance measure (4.6) and Friedman's is the first term in the squared difference. While the class conditional probability is used in our relevance measure, its expectation is used in Friedman's. This difference is driven by two different objectives: in the case of Friedman's, the goal is to seek a dimension along which the expected variation of $P(j|\mathbf{x})$ is maximized, whereas in our case we seek a dimension that minimizes the difference between the class probability distribution for a given query and its conditional expectation along that dimension (4.5).

Another fundamental difference is that the machete/scythe methods, like recursive partitioning, employ a greedy peeling strategy that removes a subset of data points permanently from further consideration. As a result, changes in an early split, due to any variability in parameter estimates, can have a significant impact on later splits, thereby producing different terminal regions. This makes predictions highly sensitive to the sampling fluctuations associated with the random

nature of the process that produces the training data, thus leading to high variance predictions. In contrast, our technique employs a "patient" averaging strategy that takes into account not only the test point $\mathbf{x}_0$ itself, but also its K_0 nearest neighbors. As such, the resulting relevance estimates (4.6) are in general more robust and have the potential to reduce the variance of the estimates, as formally shown in Section 4.6 and demonstrated in our experiments.

4.4 Estimation

Here we discuss how to estimate the unknown quantities involved in our feature relevance measure (4.5). In particular, since both $P(j|\mathbf{z})$ and $\overline{P}(j|x_i = z_i)$ in (4.5) are unknown, we must estimate them using the training data

$$\{(\mathbf{x}_n, y_n)\}_{n=1}^N$$

in order for the relevance measure (4.6) to be useful in practice. Here $y_n \in \{1, \cdots, J\}$. The quantity $P(j|\mathbf{z})$ is estimated by considering a neighborhood $N_1(\mathbf{z})$ centered at $\mathbf{z}$:

$$\hat{P}(j|\mathbf{z}) = \frac{\sum_{n=1}^N 1(\mathbf{x}_n \in N_1(\mathbf{z}))1(y_n = j)}{\sum_{n=1}^N 1(\mathbf{x}_n \in N_1(\mathbf{z}))}, \tag{4.11}$$

where $1(\cdot)$ is an indicator function such that it returns 1 when its argument is true, and 0 otherwise.

To compute $\overline{P}(j|x_i = z) = E[P(j|\mathbf{x})|x_i = z]$, we introduce an additional variable g_j such that

$$g_j|\mathbf{x} = \begin{cases} 1 & \text{if } y = j \\ 0 & \text{otherwise} \end{cases}$$

where $j = 1, \cdots, J$. We then have $P(j|\mathbf{x}) = E[g_j|\mathbf{x}]$, from which it is not hard to show that

$$\overline{P}(j|x_i = z) = E[g_j|x_i = z].$$

However, since there may not be any data at $x_i = z$, the data from the neighborhood of z along dimension i are used to estimate $E[g_j|x_i = z]$, a strategy suggested in [Fri94]. In detail, by noticing $g_j = 1(y = j)$ the estimate can be computed from

$$\hat{\overline{P}}(j|x_i = z_i) = \frac{\sum_{\mathbf{x}_n \in N_2(\mathbf{z})} 1(|x_{ni} - z_i| \leq \Delta_i) 1(y_n = j)}{\sum_{\mathbf{x}_n \in N_2(\mathbf{z})} 1(|x_{ni} - z_i| \leq \Delta_i)}, \tag{4.12}$$

where $N_2(\mathbf{z})$ is a neighborhood centered at $\mathbf{z}$ (larger than $N_1(\mathbf{z})$), and the value of Δ_i is chosen so that the interval contains a fixed number L of points:

$$\sum_{n=1}^{N} 1(|x_{ni} - z_i| \leq \Delta_i) 1(\mathbf{x}_n \in N_2(\mathbf{z})) = L. \tag{4.13}$$

Using the estimates in (4.11) and in (4.12), we obtain an empirical measure of the relevance (4.6) for each input variable i.

Given a test point $\mathbf{x}_0$, and input parameters K_0, K_1, K_2, L, K, and c:

1. Initialize $\mathbf{w}$ in (6.4) to 1;
2. Compute the K_0 nearest neighbors of $\mathbf{x}_0$ using the weighted distance metric (6.4);
3. For each dimension i, $i = 1, \ldots, q$, compute relevance estimate $\bar{r}_i(\mathbf{x}_0)$ (4.6) through Equations (4.11) and (4.12);
4. Update $\mathbf{w}$ according to (4.7) or (4.8);
5. Iterate steps 2, 3, and 4 (zero and five times in our experiments);
6. At completion, use $\mathbf{w}$, hence (4.9), for K-nearest neighbor classification at the test point $\mathbf{x}_0$.

Figure 4.2: The ADAMENN algorithm

4.5 Adaptive Metric Nearest Neighbor Algorithm

The adaptive metric nearest neighbor algorithm (ADAMENN) has six adjustable tuning parameters:

- K_0: the number of neighbors of the test point;
- K_1: the number of neighbors in $N_1(\mathbf{z})$ for estimation (4.11);
- K_2: the size of the neighborhood $N_2(\mathbf{z})$ for each of the K_0 neighbors for estimation (4.12);
- L: the number of points within the Δ intervals;
- K: the number of neighbors in the final nearest neighbor rule;
- c: the positive factor for the exponential weighting scheme (4.8).

Cross-validation can be used to determine the optimal values of the parameters. Note that K is common to all NN rules. We have considered the range of values $\{1, \cdots, 9\}$ for K in our experiments. K_0 is used to reduce the variance of the estimates; its value should be a small fraction of N. We set $K_0 = \max(0.1N, 20)$ in our experiments. Often a smaller value is preferable for K_1 to avoid biased estimates; $K_1 \in \{1, \cdots, 9\}$ in our experiments. We obtained optimal performance for small values (one or three) of parameters K and K_1 in all our experiments. K_2 and L are common to the machete and scythe algorithms described in [Fri94]. The values of K_2 and L determine the bias and variance trade-off for the estimation of $E[g_j|x_i = z]$. The way these estimates are used does not require a high accuracy. As a consequence, ADAMENN performance is basically insensitive to the values chosen for K_2 and L, provided they are not too small (close to one), nor too large (close to N). We set $K_2 = 0.15N$ and $L = K_2/2$ in our experiments. The value of c should increase as the input query moves close to the decision boundary, so that highly stretched neighborhoods will result. We chose c empirically in our experiments, and considered the range of values $\{1, \cdots, 20\}$. Different values of the c factor turned out to be optimal for different problems (5, 11, and 16).

Arguably we have introduced a few more parameters that might potentially cause overfitting. However, it is important to realize that one of the parameters (K_0) plays the role of averaging or smoothing. Because it helps reduce variance, we can afford to have a few parameters that adapt to produce modified neighborhoods having more homogeneous posterior probabilities, thereby reducing estimation bias. As a result, performance can be improved, as evidenced by the results shown in Chapter 5.

At the beginning, the estimation of the $\bar{r}_i$ values in (4.6) is accomplished by using a weighted distance metric (6.4) with $\mathbf{w}$ being initialized to 1. Then, the elements w_i of $\mathbf{w}$ are updated according to $\bar{r}_i$ values via (4.7) or (4.8). In our experiments, we tested both a linear and an exponential weighting scheme. We obtained better results using the exponential scheme, therefore we present the results for this case. The update of $\mathbf{w}$ can be iterated. At completion, the resulting $\mathbf{w}$ is plugged in (6.4) to compute nearest neighbors at the test point $\mathbf{x}_0$.

4.6 Rationale for Averaging Relevance Estimate

In this section we show more formally that averaging in (4.6) potentially reduces the overall mean-squared estimation error, thereby improving classification performance. Let $\mathbf{x}_0$ be a given query point. For each given dimension i, our goal to estimate t_i:

$$t_i(\mathbf{x}_0) = \sum_{j=1}^{J} \frac{[P(j|\mathbf{x}_0) - \overline{P}(j|x_i = x_{0i})]^2}{\overline{P}(j|x_i = x_{0i})}. \tag{4.14}$$

Let $r_i(\mathbf{x}_0, \mathbf{z})$ be the estimator as defined by Equation (4.5), where $\mathbf{z}$ is in $N(\mathbf{x}_0)$. Then the aggregated estimator

$$\bar{r}_i(\mathbf{x}_0) = E_{\mathbf{z}} r_i(\mathbf{x}_0, \mathbf{z})$$

is the average over $\mathbf{z}$ of $r_i(\mathbf{x}_0, \mathbf{z})$ in a neighborhood $N(\mathbf{x}_0)$ of $\mathbf{x}_0$.

Lemma 1 *The averaging of relevance estimate in (4.6) reduces the overall mean-squared estimation error:*

$$E_{\mathbf{x}_0}[\sum_{i=1}^{q}(t_i(\mathbf{x}_0) - \bar{r}_i(\mathbf{x}_0))^2] \leq E_{\mathbf{x}_0}[\sum_{i=1}^{q} E_{\mathbf{z}}[(t_i(\mathbf{x}_0) - r_i(\mathbf{x}_0, \mathbf{z}))^2]$$

Proof: Assume $\mathbf{x}_0$ is fixed and $t_i(\mathbf{x}_0)$ is the relevance value for dimension i at $\mathbf{x}_0$. Then the combined mean-squared error of the estimates $r_i(\mathbf{x}_0, \mathbf{z})$ for all q dimensions is

$$\sum_{i=1}^{q} E_{\mathbf{z}}[(t_i(\mathbf{x}_0) - r_i(\mathbf{x}_0, \mathbf{z}))^2] = \sum_{i=1}^{q}(t_i^2(\mathbf{x}_0) - 2t_i(\mathbf{x}_0)E_{\mathbf{z}}[r_i(\mathbf{x}_0, \mathbf{z})] + E_{\mathbf{z}}[r_i^2(\mathbf{x}_0, \mathbf{z})]). \quad (4.15)$$

Applying $E[X^2] \geq E^2[X]$ to the third term in (4.15) gives

$$\begin{aligned}
\sum_{i=1}^{q} E_{\mathbf{z}}[(t_i(\mathbf{x}_0) - r_i(\mathbf{x}_0, \mathbf{z}))^2] &\geq \sum_{i=1}^{q}(t_i^2(\mathbf{x}_0) - 2t_i(\mathbf{x}_0)E_{\mathbf{z}}[r_i(\mathbf{x}_0, \mathbf{z})] + E_{\mathbf{z}}^2[r_i(\mathbf{x}_0, \mathbf{z})]) \\
&= \sum_{i=1}^{q}(t_i(\mathbf{x}_0) - E_{\mathbf{z}}[r_i(\mathbf{x}_0, \mathbf{z})])^2 \\
&= \sum_{i=1}^{q}(t_i(\mathbf{x}_0) - \bar{r}_i(\mathbf{x}_0))^2 \quad (4.16)
\end{aligned}$$

Integrating both sides of (4.16) over the joint distribution of $t_i(\mathbf{x}_0)$ and $\mathbf{x}_0$, we can conclude that the mean-squared error of $\bar{r}_i(\mathbf{x}_0)$ is lower than the mean-squared error of $r_i(\mathbf{x}_0, \mathbf{z})$ averaged over $\mathbf{z}$. This concludes our proof. ■

We note that $\bar{r}_i(\mathbf{x}_0)$ is a function of both $\mathbf{x}_0$ and the probability distribution P from which the training data are drawn. Of course, our estimate (4.6) is not $E_{\mathbf{z}} r_i(\mathbf{x}_0, \mathbf{z})$. Instead, it follows the distribution that allocates $1/K$ to each $\mathbf{z} \in N(\mathbf{x}_0)$. The gain in error reduction depends on how unequal the two sides of (4.16) are. This is in direct analogy to improvement in performance that can be achieved by bagging predictors [Bre96].

4.7 Discussion

While it is difficult in general to answer the question of convergence of ADAMENN, we can gain some insight into the issue in a way similar to that used in [HT96a]. Let Σ be a diagonal matrix, where the diagonal entries are defined by (4.8). Our weighted distance metric (4.9) can be written as $\sqrt{(\mathbf{x}-\mathbf{x}_0)^t\Sigma(\mathbf{x}-\mathbf{x}_0)}$. At each step we take a spherical neighborhood of the query point, compute the metric Σ, and transform the feature vectors through $\mathbf{x}^{(i+1)} = \Sigma^{1/2}\mathbf{x}^{(i)}$. Thus, the effective metric for the original feature vectors is $\Sigma_1^{1/2}\Sigma_2^{1/2}\cdots\Sigma_{i-1}^{1/2}\Sigma_i\Sigma_{i-1}^{1/2}\cdots\Sigma_2^{1/2}\Sigma_1^{1/2}$, where Σ_i is the metric estimated at the ith iteration. It follows that the fixed point of the transformation satisfies $\Sigma = I$. That is, we have a fixed point when Σ is the identity matrix in the transformed space. In such a space, all features have equal relevance. In practice, however, Σ could only become proportional to the identity matrix, since the diagonal entries (weights) are normalized. Therefore, the stopping criterion is met when the diagonal entries of Σ are all equal to $1/q$ in the transformed space, in which case we still have equal relevance among the features.

As discussed in Section 4.3 we have considered estimating feature relevance along each individual dimension, one at the time. A potential extension is to consider additional derived variables (features) for local relevance estimate, thereby contributing to the distance calculation. When the derived features are more informative, huge gains may be expected. On the other hand, if they are not informative enough, they may cause classification performance to degrade since they add to the dimensionality count. The challenge is to be able to have a mechanism that computes such informative derived features efficiently.

The local nature of adaptation performed in ADAMENN makes the idea of considering

derived variables particularly attractive. Derived variables can be customized to a specific query point where prediction is to be made, thereby defining different derived variables for different query points. Here, we consider derived variables d that are linear combinations of the input features x_i:

$$d = g(\mathbf{x}|\mathbf{x}_0) = \sum_{i=1}^{q} a_i(\mathbf{x}_0) x_i. \tag{4.17}$$

The challenge is to compute the values of the coefficients $a_i(\mathbf{x}_0)$ that make d highly relevant. One possibility consists of using ADAMENN to determine the relevant input features at $\mathbf{x}_0$ and combine them as in (4.17) by using as coefficients the corresponding weight values (4.8). It is reasonable to expect, in fact, that the locally relevant features might be correlated, and by combining them we may obtain a highly relevant derived variable. The relevant features to be included in (4.17) are those whose relevance is above the average relevance value computed across all input variables. Once d has been derived, we run ADAMENN including d as an input variable.

Since the above procedure is computationally costly, we may achieve an approximation by partitioning the input space into regions and producing for each region good discriminant linear combination of variables. We can rely on discriminant analysis for producing such derived variables, either by considering all classes (represented in current region) collectively, or by isolating one class at a time. Then, for a given query point $\mathbf{x}_0$, the derived variables to be considered are the ones pre-computed for the region that has the largest overlap with a neighborhood region centered at $\mathbf{x}_0$.

Chapter 5

ADAMENN: Experimental Evaluation

This chapter presents an extensive experimental evaluation comparing ADAMENN with well-known techniques in the literature, using both simulated and real data.

5.1 Methods Compared

We use both simulated and real data to evaluate the performance of ADAMENN and compare it with several competing methods. The simulated data experiments allow us to reliably predict the strengths and limitations of algorithms because the precise nature of the problem the algorithms are facing is known. We compare the following classification approaches:

- ADAMENN-adaptive metric nearest neighbor described in Figure 4.2 (one iteration), coupled with the exponential weighting scheme (4.8);
- i-ADAMENN-adaptive metric nearest neighbor with five iterations;
- Simple K-NN method using the Euclidean distance measure;

- C4.5 decision tree method [Qui93];
- Machete [Fri94] (see Chapter 3);
- Scythe [Fri94] (see Chapter 3);
- DANN-discriminant adaptive nearest neighbor classification [HT96a] (see Chapter 3);
- i-DANN-discriminant adaptive nearest neighbor classification [HT96a] with five iterations.

We focus the experimental evaluation on locally adaptive techniques, since ADAMENN belongs to this category. While enhanced performance is expected by boosting C4.5, all adaptive algorithms considered here could also be boosted. We do not include any boosting method in this study.

In all the experiments, the features are first normalized over the training data to have zero mean and unit variance, and the test data features are normalized using the corresponding training mean and variance.

5.1.1 Choice of Tuning Parameters

Procedural parameters for each method were determined empirically through cross-validation over the training data. Details for setting the parameters of ADAMENN are provided in Section 4.5. We have considered the range of values $\{1, \ldots, 9\}$ for K in the K-NN method.

The machete and scythe algorithms have the same L and K input parameters as ADAMENN. We apply the same guidelines described in Section 4.5 to set them. In addition, for the machete and scythe techniques we need to specify the rate at which the sample size decreases in the sequence

of regions produced by the successive splitting (α parameter in [Fri94]). We set $\alpha = 0.5$ in our experiments.

The DANN algorithm has two adjustable parameters: the number of nearest neighbors of the query point for estimation of the metric, and the number of neighbors in the final nearest neighbor rule. We set the first parameter as K_0 for ADAMENN, i.e. equal to $\max(0.1N, 20)$. For the second parameter, we again consider the range of values $\{1, \ldots, 9\}$.

5.2 Experiments on Simulated Data

For all simulated data, 20 independent training samples (of size N) were generated. For each of these, an additional independent test sample consisting of 500 observations was generated. These test data were classified by each competing method using the respective training data set. Error rates computed over all 10,000 such classifications are reported in Table 5.1.

5.2.1 The Problems

1. This problem is taken from [Fri94], and designed to be favorable to the adaptive methods (ADAMENN/DANN/scythe/machete/C4.5), and unfavorable to the regular K-NN procedure. There are $q = 10$ input features, $N = 200$ training data, and $J = 2$ classes. The data for the first class were generated from a standard normal distribution $\mathbf{x}_n \sim N(\mathbf{0}, \mathbf{1})$. The data for the second class were also generated from a normal distribution $\mathbf{x}_n \sim N(\mathbf{m}, \mathbf{C})$, with the coordinate mean values and covariance matrix given by

$$\{m_i = \sqrt{i}/2\}_{i=1}^{q}, \;\; \mathbf{C} = diag\{1/\sqrt{i}\}_{i=1}^{q}.$$

Although all input variables are relevant, the ones with higher coordinate number i are more so. Also, since only the diagonal elements of the covariance matrix are non zeros, much of the discriminating information is axis oriented. The first column of Table 5.1 shows the results for the eight methods under comparison, with standard deviations: 1.86, 1.36, 1.98, 1.78, 1.19, 1.62, 1.16 and 1.18 respectively. i-DANN had the lowest error rate, with DANN exhibiting similar performance. As expected, the K-NN procedure had the poorest performance for this problem.

2. This problem is adapted from [HT96a], and consists of four dimensional spheres with 6 noise features. There are $q = 10$ input features, $N = 200$ training data, and $J = 2$ classes. The last 6 features are noise variables, with standard Gaussian distributions, independent of each other and the class membership. The data for both classes are generated from a standard normal distribution. The data for class one have the property that the radius, computed from the first four features, is greater than 1.85 while the data for class two do not have such restriction. Class one basically surrounds class two in the subspace spanned by the first four features. Results are shown in the second column of Table 5.1. The standard deviations are: 2.30, 2.83, 2.73, 1.56, 2.44, 2.59, 3.17, and 2.26 respectively. C4.5 is by far the best performer in this case. Machete gives the second best performance, with i-DANN and i-ADAMENN being close to it. K-NN performs very poorly on this problem.

3. This example is also taken from [Fri94]. It is designed to be more favorable to the K-NN procedure, since all the input variables have the same global relevance. As before there are $q = 10$ input features and $J = 2$ classes, but $N = 500$ training data. The data for both classes are generated

from a standard normal distribution $\mathbf{x}_n \sim N(\mathbf{0}, \mathbf{1})$, and the classes are defined by

$$\sum_{i=1}^{10} x_i^2 \leq 9.8 \Rightarrow class1, \quad otherwise \Rightarrow class2.$$

The third column of Table 5.1 shows the results for this example, with the standard deviations: 2.09, 2.01, 2.35, 2.74, 2.71, 1.97, 2.36 and 1.73 respectively. The K-NN procedure exhibits a more competitive performance with the adaptive techniques, even though it still has the worst error rate. i-DANN shows the best performance for this problem. DANN gives a similar result, with C4.5 being the closest to it. Note that for this example, the performance of ADAMENN doesn't improve by performing five iterations.

4. This example is again taken from [Fri94]. It is constructed so that all input variables have equal local relevance everywhere in the input space. However, there is a single direction in the space that contains all the discriminant information. There are $q = 10$ input features, $N = 200$ training data, and $J = 2$ classes. The data for both classes are generated from a standard normal distribution $\mathbf{x}_n \sim N(\mathbf{0}, \mathbf{1})$, and the classes are defined by

$$\sum_{i=1}^{10} x_i \leq 0 \Rightarrow class1, \quad otherwise \Rightarrow class2.$$

Results are shown in the fourth column of Table 5.1. The standard deviations are: 2.32, 1.83, 1.62, 2.06, 1.92, 1.99, 2.20 and 1.86. i-DANN gives the best performance, with DANN being very close to it. K-NN performs well because all variables are equally locally relevant everywhere. ADAMENN and i-ADAMENN come close to it, showing that with our adaptive method we don't loose much when all variables are equally relevant. For this example, the performance of ADAMENN improves only slightly by performing 5 iterations. C4.5 is the worst performer in this case.

5. This problem is adapted from [HT96a]. There are $q = 2$ input features, $N = 200$ training data, and $J = 2$ classes. Each class contains six spherical bivariate normal subclasses, having standard deviation 0.25. The means of the 12 subclasses are chosen at random without replacement from the integers $[1, 2, \ldots, 5] \times [1, 2, \ldots, 5]$. For each class, data are evenly drawn from each of the six normal subclasses. The fifth column of Table 5.1 shows the results for this problem, with standard deviations: 0.83, 0.78, 0.86, 6.01, 0.82, 0.76, 1.11 and 1.20, respectively. ADAMENN, i-ADAMENN and scythe show the best performance for this problem. K-NN and machete give similar results. C4.5 is again the worst performer.

6. This problem is taken from [HT96a]. There are $q = 2$ input features, $N = 200$ training data, and $J = 4$ classes. Each class contains three spherical bivariate normal subclasses, having standard deviation 0.25. As in the previous example, the means of the 12 subclasses are chosen at random without replacement from the integers $[1, 2, \ldots, 5] \times [1, 2, \ldots, 5]$. For each class, data are evenly drawn from each of the three normal subclasses. Results are shown in the sixth column of Table 5.1. The standard deviations are: 1.46, 1.37, 1.19, 8.57, 5.04, 5.36, 1.69 and 1.75, respectively. For this problem ADAMENN, i-ADAMENN and K-NN give the best performance. DANN shows a similar result. Again, the worst performer is C4.5.

7. This problem is also taken from [HT96a]. There are $q = 10$ input features, $N = 200$ training data, and $J = 4$ classes. The data for this problem are generated as in the previous example, but augmented with eight predictors having independent standard Gaussian distributions. They serve as noise. The seventh column of Table 5.1 shows the results, with standard deviations: 3.93, 3.66, 8.02, 16.84, 5.31, 4.96, 8.31 and 7.74, respectively. ADAMENN is by far the best performer in this

case, with only i-ADAMENN coming close to it. K-NN gives the worst performance in this case.

5.2.2 Results

Table 5.1 shows that, for each method, there is at least one example for which it has the best performance, or close to the best. Therefore, it seems natural to ask the question of robustness. That is, how well a particular method m performs on average in situations that are most favorable to other procedures. Following Friedman [Fri94], we capture robustness by computing the ratio b_m of its error rate e_m and the smallest error rate over all methods being compared in a particular example:

$$b_m = e_m / \min_{1 \leq k \leq 8} e_k.$$

Thus, the best method m^* for that example has $b_{m^*} = 1$, and all other methods have larger values $b_m \geq 1$, for $m \neq m^*$. The larger the value of b_m, the worse the performance of method m is in relation to the best one for that example, among the methods being compared. The distribution of the b_m values for each method m over all the examples, therefore, seems to be a good indicator concerning its robustness. For example, if a particular method has an error rate close to the best in every problem, its b_m values should be densely distributed around the value 1. Any method whose b value distribution deviates from this ideal distribution reflect its lack of robustness.

Figure 5.1 plots the distribution of b_m for each method over the seven simulated data sets. The dark area represents the lower and upper quartiles of the distribution that are separated by the median. The outer vertical lines show the entire range of values for the distribution. It is clear that the most robust method over the simulated data is i-ADAMENN. In 4/7 of the data its error rate was no worse than 33% higher than the best error rate. In the worst case it was 87%. In contrast, C4.5

Table 5.1: Average classification error rates for simulated data.

	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6	Ex7
ADAMENN	9.9	23.9	33.7	20.8	**2.4**	**3.3**	**12.8**
i-ADAMENN	8.3	23.1	33.7	20.3	**2.4**	**3.3**	14.2
K-NN	14.7	33.9	36.1	18.1	2.5	**3.3**	50.7
C4.5	10.3	**14.6**	30.6	30.1	25.5	31.7	38.2
Machete	7.1	21.7	33.0	25.7	2.6	14.5	20.1
Scythe	7.9	25.6	32.7	22.2	**2.4**	21.3	38.1
DANN	6.2	25.3	26.7	13.3	2.8	4.2	37.6
i-DANN	**5.3**	22.8	**25.4**	**13.2**	3.1	6.1	26.7

has the worst distribution, where the corresponding numbers are 128% and 962%.

DANN and i-DANN performed well in examples 1, 3 and 4, where the data were generated from Gaussian distributions. This might be attributed to the fact that the distance metric computed by DANN approximates the weighted *Chi-squared* distance (4.3), only when class densities are Gaussian and have the same covariance matrix. This may also explain DANN's performance degradation in those examples where data do not follow Gaussian distributions or are corrupted by noise.

5.3 Experiments on Real Data

While simulated data are informative for comparison studies, it is highly likely that artificially constructed examples will not correspond to situations that are likely to occur in practice. Thus, in this section we examine the performance of the competing classification methods using real world data. One of the advantages of real data is that they are generated without any knowledge of the classification procedures that it will be used to test.

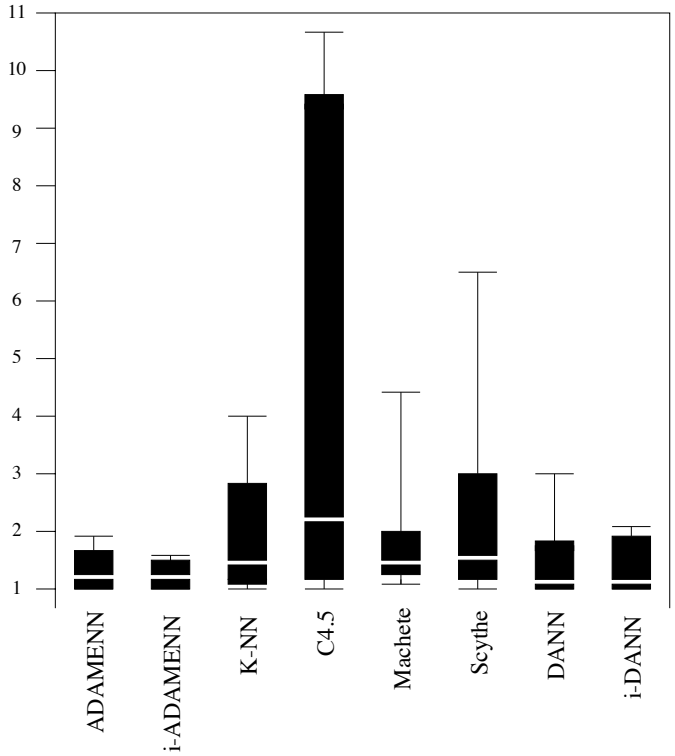

Figure 5.1: Performance distributions for simulated data.

In our experiments we used nine different real data sets. The Iris, Sonar, Vowel, Glass, Segmentation, Letter, Liver, and Lung data are taken from UCI Machine Learning Repository at http://www.cs.uci.edu/∼mlearn/MLRepository.html. The Image data are obtained from MIT Media Lab at ftp://whitechapel.media.mit.edu/pub/VisTex. If an attribute of a data set has missing values, we eliminate that feature from the entire set. For the Iris, Sonar, Glass, Liver, and Lung data we perform leave-one-out cross-validation to measure performance. For the Vowel and Image data we randomly divide the data into a training set of 200 data points and a test set consisting of the remaining data points (320 for the Vowel data and 440 for the Image data). We repeat this process 10 times independently, and report the average cross-validation error rates for these two data sets. On the Segmentation and Letter data we perform two 10-fold cross-validation. We randomly divide

the data into 10 sets of equal size and use one of them in turn as a test set and the remaining nine as a training set. We repeat this process two times independently and report the two 10-fold cross-validation error rates for these two data sets. Table 5.2 shows the cross-validated error rates for the eight methods under consideration on the nine real data.

5.3.1 The Problems

1. **Iris data**. This data set consists of $q = 4$ measurements made on each of $N = 100$ iris plants of $J = 2$ species. The two species are iris versicolor and iris virginica. The problem is to classify each test point to its correct species based on the four measurements. The results on this data set are shown in the first column of Table 5.2.

2. **Sonar data**. This data set consists of $q = 60$ frequency measurements made on each of $N = 208$ data of $J = 2$ classes ("mines" and "rocks"). The problem is to classify each test point in the 60-dimensional feature space to its correct class. The results on this data set are shown in the second column of Table 5.2.

3. **Vowel data**. This example has $q = 10$ measurements and $J = 11$ classes. There are $N = 528$ samples in this example. Results are shown in the third column of Table 5.2, having standard deviations: 2.82, 3.06, 2.56, 3.68, 2.82, 2.30, 3.06 and 2.93 respectively.

4. **Glass data**. This data set consists of $q = 9$ chemical attributes measured for each of $N = 214$ data of $J = 6$ classes. The problem is to classify each test point in the 9-dimensional space to its correct class. Results are shown in the fourth column of Table 5.2.

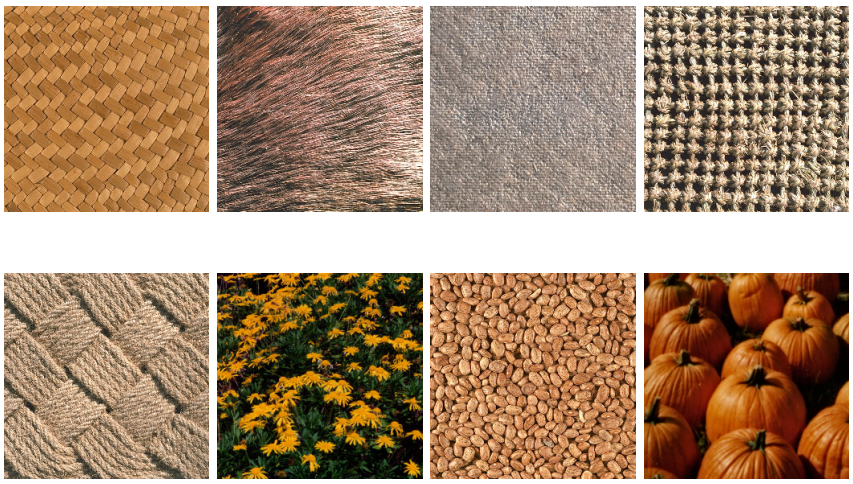

Figure 5.2: Sample images taken from the Image database.

5. **Image data**. This data set consists of 40 texture images that are manually classified into 15 classes. Each of these images is then cut into 16 non-overlapping images of 128×128, giving rise to a total of 640 images in the database. Sample images are shown in Figure 5.2. The number of images in each class varies from 16 to 80. The images in this database are represented by $q = 16$ dimensional feature vectors (8 Gabor filters: 2 scales and 4 orientations). The mean and the standard deviation of the magnitude of the transform coefficients are used as feature components, after being normalized by the standard deviations of the respective features, over the entire set of images in the database. Results are shown in the fifth column of Table 5.2. The standard deviations are: 0.78, 1.31, 1.21, 3.0, 1.69, 1.57, 2.23 and 3.35 respectively.

6. **Segmentation data**. This data set consists of images that were drawn randomly from a database of 7 outdoor images. The images were hand segmented by the creators of the database to classify each pixel. Each image is a region. There are $J = 7$ classes, each of which has 330 instances. Thus,

there are $N = 2,310$ images in the database. These images are represented by $q = 19$ real valued attributes. Results are shown in the sixth column of Table 5.2. The standard deviations are: 0.91, 1.07, 1.27, 1.15, 1.17, 1.28, 0.91 and 1.10, respectively.

7. **Letter Image Recognition data**. This data set consists of $q = 16$ numerical attributes and $J = 26$ classes. The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. Sample images are shown in Figure 5.3. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. Results are shown in the seventh column of Table 5.2. The standard deviations are: 0.78, 0.71, 0.91, 0.88, 0.87, 0.87, 1.14 and 0.86, respectively.

8. **Liver data**. This data set has $q = 6$ attribute values regarding blood test results and drinking habits. There are $J = 2$ classes, establishing the liver disorder condition, and $N = 345$ samples. The results on this data set are shown in the eighth column of Table 5.2.

9. **Lung data**. This data set has $q = 56$ attribute values, $J = 3$ classes, and $N = 32$ samples. The results on this data set are shown in the ninth column of Table 5.2.

5.3.2 Results

Table 5.2 shows that ADAMENN achieved the best performance in 5/9 of the real data sets, followed closely by i-ADAMENN. For three of the remaining data sets, ADAMENN has

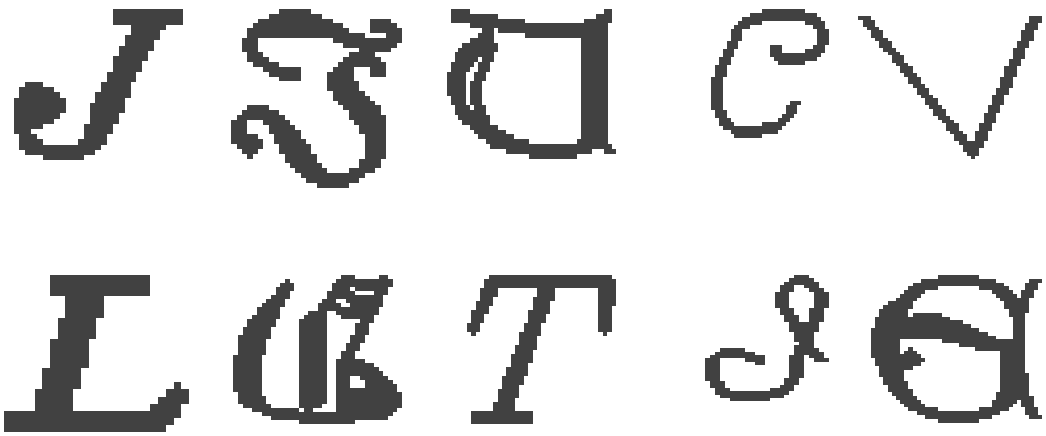

Figure 5.3: Sample letter images.

the second best performance. As shown in Figure 5.4, the spread of the error distribution for ADAMENN is narrow and close to 1. The spread for i-ADAMENN has a similar behavior. The results clearly demonstrate that they obtained the most robust performance over these data sets. Similar characteristics were also observed for the two methods over the simulated data sets. This could be attributed to the fact that local feature relevance estimate in ADAMENN is conducted over regions in the feature space instead of using individual points, as is done in machete and scythe [Fri94]. This observation is corroborated by our discussion in Section 4.6.

5.4 Bias and Variance Calculations

For a two-class problem with $P(Y = 1|\mathbf{x}) = p(\mathbf{x})$, we compute a nearest neighborhood at a query $\mathbf{x}_0$ and find the nearest neighbor $\mathbf{X}$ having class label $Y(\mathbf{X})$ (random variable). The estimate of $p(\mathbf{x}_0)$ is $Y(\mathbf{X})$. The bias and variance of $Y(\mathbf{X})$ are: $Bias = Ep(\mathbf{X}) - p(\mathbf{x}_0)$ and

Table 5.2: Average classification error rates for real data.

	Iris	Sonar	Vowel	Glass	Image	Seg	Letter	Liver	Lung
ADAMENN	**3.0**	9.1	**10.7**	**24.8**	5.2	**2.4**	5.1	30.7	**40.6**
i-ADAMENN	5.0	9.6	10.9	**24.8**	5.2	2.5	5.3	30.4	**40.6**
K-NN	6.0	12.5	11.8	28.0	6.1	3.6	6.9	32.5	50.0
C4.5	8.0	23.1	36.7	31.8	21.6	3.7	16.4	38.3	59.4
Machete	5.0	21.2	20.2	28.0	12.3	3.2	9.1	**27.5**	50.0
Scythe	4.0	16.3	15.5	27.1	**5.0**	3.3	7.2	**27.5**	50.0
DANN	6.0	**7.7**	12.5	27.1	12.9	2.5	**3.1**	30.1	46.9
i-DANN	6.0	9.1	21.8	26.6	18.1	3.7	6.1	27.8	**40.6**

$Var = Ep(\mathbf{X})(1 - Ep(\mathbf{X}))$, where the expectation is computed over the distribution of the nearest neighbor $\mathbf{X}$ [HT96a].

We performed simulations to estimate the bias and variance of ADAMENN, K-NN, DANN and Machete on the following two-class problem. There are $q = 2$ input features and 180 training data. Each class contains three spherical bivariate normal subclasses, having standard deviation 0.75. The means of the 6 subclasses are chosen at random without replacement from the integers $[1, 2, \ldots, 8] \times [1, 2, \ldots, 8]$. For each class, data are evenly drawn from each of the normal subclasses. Figures 5.5 and 5.6 show the bias and variance estimates from each method at locations $(5, 5, 0, \cdots, 0)$ and $(2.3, 7, 0, \cdots, 0)$, as a function of the number of noise variables over five independently generated training sets. Figure 5.4 shows the corresponding mean squared errors. Here the noise variables have independent standard Gaussian distributions. The true probability of class 1 for $(5, 5, 0, \cdots, 0)$ and $(2.3, 7, 0, \cdots, 0)$ are 0.943 and 0.747, respectively.

The four methods have similar variance, since they all use three neighbors for classification. While the bias of K-NN and DANN increases with increasing number of noise variables,

ADAMENN retains a low bias by averaging out noise.

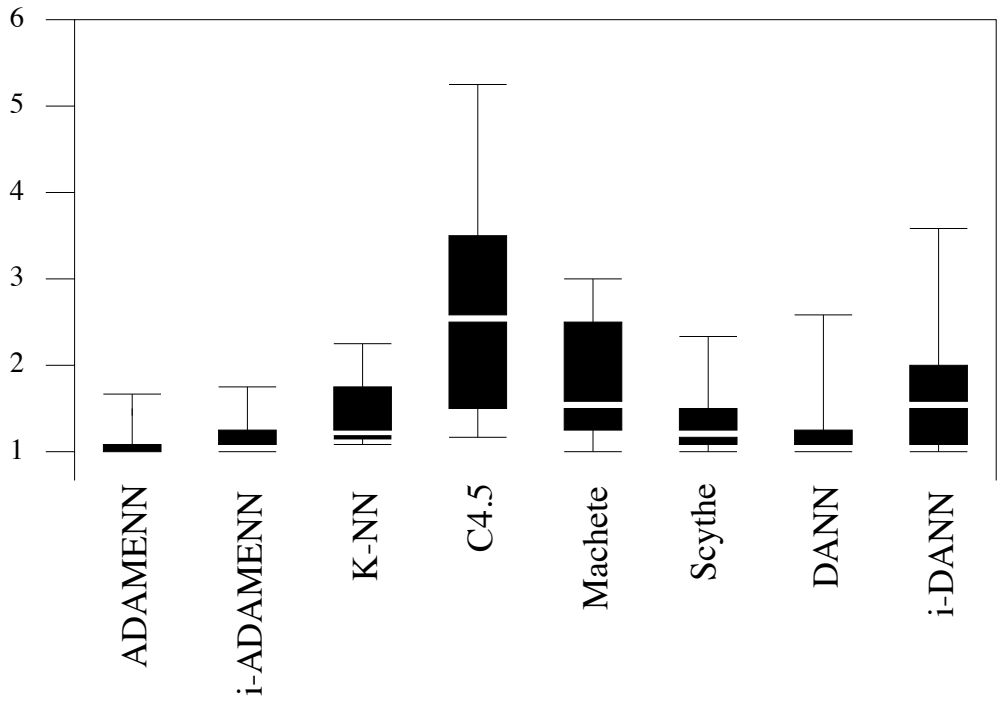

Figure 5.4: Performance distributions for real data.

5.5 Summary

We have presented an adaptive nearest neighbor method for effective pattern classification. This method estimates a flexible metric for producing neighborhoods that are elongated along less relevant feature dimensions and constricted along most influential ones. As a result, the class conditional probabilities are more homogeneous in the modified neighborhoods. The experimental results using both simulated and real data show clearly that the ADAMENN algorithm can potentially improve the performance of K-NN and recursive partitioning methods in some classification problems, especially when the relative influence of input features changes with the location of the

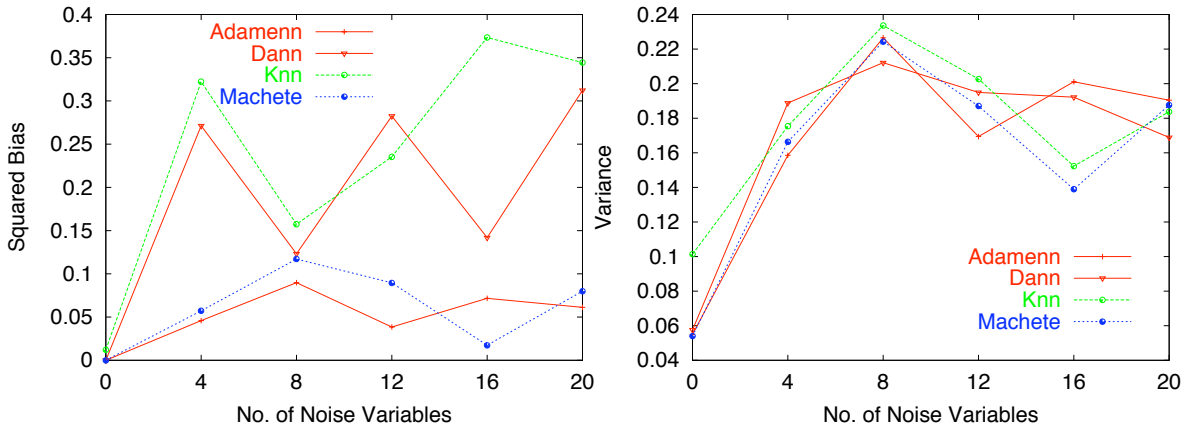

Figure 5.5: Bias and variance estimates at location $(5, 5, 0, \ldots, 0)$.

query to be classified in the input feature space. The results are also in favor of ADAMENN over other adaptive methods such as machete and DANN. It is interesting to note that our work can potentially serve as a general framework upon which to develop a unified adaptive metric theory that encompasses both Friedman's work and that of Hastie and Tibshirani.

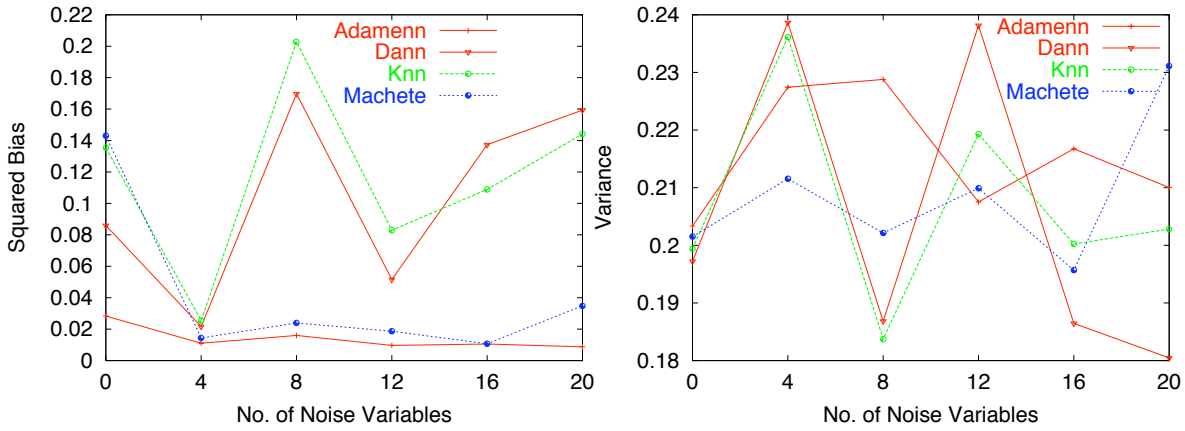

Figure 5.6: Bias and variance estimates $(2.3, 7, 0, \ldots, 0)$.

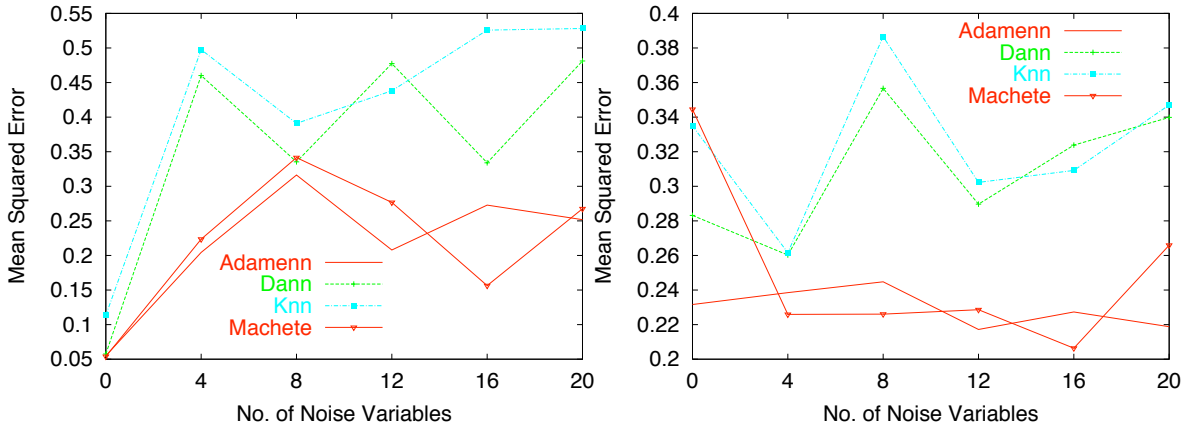

Figure 5.7: Mean Squared Errors for the Bias and Variance estimates in Figures 5.5 and 5.6.

Chapter 6

Local Flexible Metric Classification using Support Vector Machines

The major limitation of the ADAMENN algorithm concerns efficiency issues. The lazy learning approach employed by ADAMENN requires a considerable amount of on-line computation, which makes it difficult for such technique to scale up to large data sets. In this chapter we discuss a novel locally adaptive metric classification method which, although still founded on a query based weighting mechanism, computes off-line the information relevant to define local weights. The new technique computes a locally flexible metric by means of support vector machines. The maximum margin boundary found by the SVM is used to determine the most discriminant direction over the query's neighborhood. Such direction provides a local weighting scheme for input features.

6.1 Introduction

The lazy learning approach used by ADAMENN, while appealing in many ways, requires a considerable amount of on-line computation, which makes it difficult for such technique to scale up to large data sets. The feature weighting scheme it introduces, in fact, is query based and is applied on-line when the test point is presented to the lazy learner. In this chapter we discuss a locally adaptive metric classification method which, although still founded on a query based weighting mechanism, computes off-line the information relevant to define local weights.

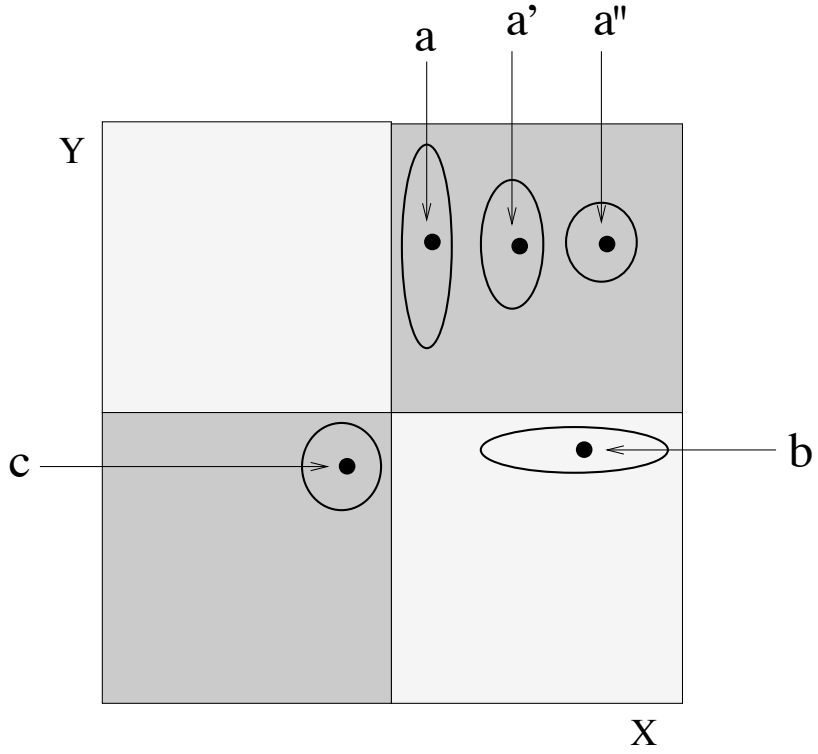

Figure 6.1: Amount of elongation-constriction decays as the query moves further from the boundary vicinity.

Our technique uses support vector machines (SVMs) as a guidance for the process of defining a local flexible metric [DG01b, DG02]. SVMs have been successfully used as a classification tool in a variety of areas [LJB+95, OFG97, Joa98, PV98, JH98, VCC99, BGL+00, Joa02,

DPV+02], and the maximum margin boundary they provide has been proved to be optimal in a structural risk minimization sense. The solid theoretical foundations that have inspired SVMs convey desirable computational and learning theoretic properties to the SVM's learning algorithm, and therefore SVMs are a natural choice for seeking local discriminant directions between classes.

The solution provided by SVMs allows to determine locations in input space where class conditional probabilities are likely to be not constant, and guides the extraction of local information in such areas. This process produces highly stretched neighborhoods along boundary directions when the query is close to the boundary. As a result, the class conditional probabilities tend to be constant in the modified neighborhoods, whereby better classification performance can be achieved. The amount of elongation-constriction decays as the query moves farther from the boundary vicinity. This phenomenon is exemplified in Figure 6.1 by queries a, a' and a''. In the next chapter we present experimental evidence of the accuracy achieved by means of this local weighting scheme.

The sparse solution given by SVMs also provides principled guidelines to efficiently set the input parameters of our technique. This is a major advantage over the ADAMENN technique, which has a competitive behavior but requires six tunable input parameters.

Furthermore, the technique proposed here speeds up the classification process since it computes off-line the information relevant to define local weights, and applies the nearest neighbor rule only once, whereas ADAMENN applies it at each point within a region centered at the query. Indeed, the major strength of our technique is that is capable of providing local feature weightings using a global decision scheme, specifically the SVM boundary. This mechanism allows an off-line computation of the relevant information to define weights, leaving only local refinements to an on-

line stage. This results in a method which is more efficient than current local adaptive techniques for nearest neighbor classification [Fri94, HT96a, DPG00a, DPG00b], which act iteratively on the computation of neighborhoods.

6.2 Feature Weighting

The maximum margin boundary found by the SVM is used here to determine local discriminant directions over query's neighborhoods. The normal direction to local decision boundaries identifies the orientation along which data points between classes are well separated. The gradient vector computed at points on the boundary allows us to capture such information, and to use it for measuring local feature relevance and weighting features accordingly. Formally, the definition of our weighting scheme proceeds as follows.

SVMs classify patterns according to the $sign(f(\mathbf{x}))$, where $f(\mathbf{x})$ is defined as in equation (2.35). Since SVMs are well suited for two class classification problems, we restrict the discussion to such case, i.e., we assume that class labels $y \in \{-1, 1\}$. Clearly, in the general case of a non-linear feature mapping ϕ, the SVM classifier gives a non-linear boundary $f(\mathbf{x}) = 0$ in input space. The gradient vector $\mathbf{n_d} = \nabla_{\mathbf{d}} f$, computed at any point $\mathbf{d}$ of the level curve $f(\mathbf{x}) = 0$, gives the perpendicular direction to the decision boundary in input space at $\mathbf{d}$. As such, the vector $\mathbf{n_d}$ identifies the orientation in input space on which the projected training data are well separated, locally over $\mathbf{d}$'s neighborhood. Therefore, the orientation given by $\mathbf{n_d}$, and any orientation close to it, is highly informative for the classification task at hand, and we can use such information to define a local measure of feature relevance.

Let $\mathbf{x}_0$ be a query point whose class label we want to predict. Suppose $\mathbf{x}_0$ is close to the

boundary, which is where class conditional probabilities become locally non uniform, and therefore estimation of local feature relevance becomes crucial. Let $\mathbf{d}$ be the closest point to $\mathbf{x}_0$ on the boundary $f(\mathbf{x}) = 0$: $\mathbf{d} = \arg\min_{\mathbf{p}} \|\mathbf{x}_0 - \mathbf{p}\|$, subject to the constraint $f(\mathbf{p}) = 0$. Then we know that the gradient $\mathbf{n_d}$ identifies a direction along which data points between classes are well separated.

As a consequence, the subspace spanned by the orientation $\mathbf{n_d}$ intersects the decision boundary and contains changes in class labels. Therefore, when applying a nearest neighbor rule at $\mathbf{x}_0$, we desire to stay close to $\mathbf{x}_0$ along the $\mathbf{n_d}$ direction, because that is where it is likely to find points similar to $\mathbf{x}_0$ in terms of the class conditional probabilities. Distances should be constricted (large weight) along $\mathbf{n_d}$ and along directions close to it, thus excluding from $\mathbf{x}_0$'s neighborhood points along $\mathbf{n_d}$ that are far from $\mathbf{x}_0$. The farther we move from the $\mathbf{n_d}$ direction, the less discriminant the correspondent orientation becomes. This means that class labels are likely not to change along those orientations, and distances should be elongated (small weight), thus including in $\mathbf{x}_0$'s neighborhood points which are likely to be similar to $\mathbf{x}_0$ in terms of the class conditional probabilities.

This principle is in analogy with a local linear discriminant analysis approach. In fact, the orientation of the gradient vector identifies the direction, locally at the query point, on which the projected training data are well separated. This property guides the process of generating modified neighborhoods with homogeneous class conditional probabilities.

Formally, we can measure how close a direction $\mathbf{t}$ is to $\mathbf{n_d}$ by considering the dot product $\mathbf{n_d}^T \cdot \mathbf{t}$. In particular, by denoting with $\mathbf{u}_j$ the unit vector along input feature j, for $j = 1, \ldots, q$, we

can define a measure of relevance for feature j, locally at $\mathbf{x}_0$ (and therefore at $\mathbf{d}$), as

$$R_j(\mathbf{x}_0) \equiv |\mathbf{u}_j^T \cdot \mathbf{n_d}| = |n_{\mathbf{d},j}| \tag{6.1}$$

where $\mathbf{n_d} = (n_{\mathbf{d},1}, \ldots, n_{\mathbf{d},q})^T$.

The measure of feature relevance, as a weighting scheme, can then be given by

$$w_j(\mathbf{x}_0) = (R_j(\mathbf{x}_0))^t / \sum_{i=1}^{q} (R_i(\mathbf{x}_0))^t \tag{6.2}$$

where $t = 1, 2$, giving rise to linear and quadratic weightings, respectively. We propose the following exponential weighting scheme

$$w_j(\mathbf{x}_0) = exp(AR_j(\mathbf{x}_0)) / \sum_{i=1}^{q} exp(AR_i(\mathbf{x}_0)) \tag{6.3}$$

where A is a parameter that can be chosen to maximize (minimize) the influence of R_j on w_j[1]. When $A = 0$ we have $w_j = 1/q$, thereby ignoring any difference between the R_j's. On the other hand, when A is large a change in R_j will be exponentially reflected in w_j. The exponential weighting is more sensitive to changes in local feature relevance (6.1), and in general gives rise to better performance improvement. In fact, the exponential weighting scheme conveys stability to the method by preventing neighborhoods to extend infinitely in any direction. This is achieved by avoiding zero weights, which is instead allowed by the linear and quadratic weightings.

Thus, (6.3) can be used as weights associated with features for weighted distance computation

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{q} w_i (x_i - y_i)^2}. \tag{6.4}$$

[1] A is the analogous of the c parameter in equation 4.8

These weights enable the neighborhood to elongate less important feature dimensions, and, at the same time, to constrict the most influential ones. Note that the technique is *query-based* because weightings depend on the query [Aha97, AMS97].

One may be tempted to use the weights $w_j(\mathbf{x}_0)$ directly in the SVM classification, by applying the weighted distance measure (6.4) in (2.35). By doing so, we would compute the weighted distances of the query point $\mathbf{x}_0$ from all support vectors, and therefore we would employ the weights $w_j(\mathbf{x}_0)$ for global distance computation over the whole input space. On the other hand, the weights $w_j(\mathbf{x}_0)$ are based on the local (to $\mathbf{x}_0$) orientation of the decision boundary, and therefore they are meaningful for local distance computation of $\mathbf{x}_0$ from its neighbors. The weights $w_j(\mathbf{x}_0)$, in fact, convey information on how distances should be constricted or elongated locally at $\mathbf{x}_0$: we desire to achieve constricted distances along directions close to the gradient direction, and elongated distances along directions far from the gradient direction. Accordingly, a locally adaptive nearest neighbor technique allows us to take into consideration only the closest neighbors (according to the learned weighted metric) in the classification process.

6.3 Local Flexible Metric Classification based on SVMs

To estimate the closest point to the query on the boundary, we move from the query point along the input axes (in both directions) at distances proportional to a given small step (whose initial value can be arbitrarily small, and doubled at each iteration till the boundary is crossed). We stop as soon as the boundary is crossed along one of the input axes, say axis i, i.e. when a point $\mathbf{p}_i$ is reached that satisfies the condition $sign(f(\mathbf{x}_0)) \times sign(f(\mathbf{p}_i)) = -1$. Given $\mathbf{p}_i$, we can get arbitrarily close to the boundary by moving at (arbitrarily) small steps along the segment that joins

$\mathbf{p}_i$ to $\mathbf{x}_0$.

Let us denote with $\mathbf{d}_i$ the intercepted point on the boundary along direction i. We then approximate $\mathbf{n_d}$ with the gradient vector $\mathbf{n}_{\mathbf{d}_i} = \nabla_{\mathbf{d}_i} f$, computed at $\mathbf{d}_i$.

We desire that the parameter A in the exponential weighting scheme (6.3) increases as the distance of $\mathbf{x}_0$ from the boundary decreases. By using the knowledge that support vectors are mostly located around the boundary surface, we can estimate how close a query point $\mathbf{x}_0$ is to the boundary by computing its distance from the closest non bounded support vector:

$$B_{\mathbf{x}_0} = \min_{\mathbf{s}_i} \|\mathbf{x}_0 - \mathbf{s}_i\| \tag{6.5}$$

where the minimum is taken over the non bounded ($0 < \alpha_i < C$) support vectors $\mathbf{s}_i$. Following the same principle, in [AW99] the spatial resolution around the boundary is increased by enlarging volume elements locally in neighborhoods of support vectors.

Then, we can achieve our goal by setting

$$A = D - B_{\mathbf{x}_0} \tag{6.6}$$

where D is a constant input parameter of the algorithm. In our experiments we set D equal to the approximated average distance between the training points $\mathbf{x}_k$ and the boundary:

$$D = \frac{1}{N} \sum_{\mathbf{x}_k} \{\min_{\mathbf{s}_i} \|\mathbf{x}_k - \mathbf{s}_i\|\}. \tag{6.7}$$

By doing so the value of A nicely adapts to each query point according to its location with respect to the boundary. The closer $\mathbf{x}_0$ is to the decision boundary, the higher the effect of the R_j's values will be on distances computation.

Input: Decision boundary $f(\mathbf{x}) = 0$ produced by an SVM; query point $\mathbf{x}_0$ and parameter K.

1. Compute the approximated closest point $\mathbf{d}_i$ to $\mathbf{x}_0$ on the boundary;
2. Compute the gradient vector $\mathbf{n}_{\mathbf{d}_i} = \nabla_{\mathbf{d}_i} f$;
3. Set feature relevance values $R_j(\mathbf{x}_0) = |n_{\mathbf{d}_i,j}|$ for $j = 1, \ldots, q$;
4. Estimate the distance of $\mathbf{x}_0$ from the boundary as: $B_{\mathbf{x}_0} = \min_{\mathbf{s}_i} \|\mathbf{q} - \mathbf{s}_i\|$;
5. Set $A = D - B_{\mathbf{q}}$, where D is defined as in equation (6.7);
6. Set $\mathbf{w}$ according to (6.3);
7. Use the resulting $\mathbf{w}$ for K-nearest neighbor classification at the query point $\mathbf{x}_0$.

Figure 6.2: The LFM-SVM algorithm

We observe that this principled guideline for setting the parameters of our technique takes advantage of the sparseness representation of the solution provided by the SVM. In fact, for each query point $\mathbf{x}_0$, in order to compute $B_{\mathbf{x}_0}$ we only need to consider the support vectors, whose number is typically small compared to the total number of training examples. Furthermore, the computation of D's value is carried out once and off-line, since it does not depend on the query $\mathbf{x}_0$.

The resulting local flexible metric technique based on SVMs (LFM-SVM) is summarized in Figure 6.2. The algorithm has only one adjustable tuning parameter, namely the number K of neighbors in the final nearest neighbor rule. This parameter is common to all nearest neighbor classification techniques.

In [AW99], Amari and Wu improve support vector machine classifiers by modifying kernel functions. A primary kernel is first used to obtain support vectors. The kernel is then modified in a data dependent way by using the information of the support vectors: the factor that drives the transformation has larger values at positions close to support vectors. The modified kernel enlarges

the spatial resolution around the boundary so that the separability of classes is increased.

The resulting transformation depends on the distance of data points from the support vectors, and it is therefore a local transformation, but is independent of the boundary's orientation in input space. Likewise, our transformation metric depends on the distance of the query point from the support vectors; this dependence is driven by the A factor in the exponential weighting scheme (6.3), which is defined in (6.6). Moreover, since we weight features, our metric is directional, and depends on the orientation of local boundaries in input space. This dependence is driven by the measure of feature relevance (6.1), which has the effect of increasing the spatial resolution along discriminant directions around the boundary.

6.4 Weighting Features Increases the Margin

In this section we formally show that our weighting scheme increases the margin of the solution provided by the SVM. Our discussion holds for Gaussian kernels. This property explains the performance improvements achieved by our method over the SVM alone, as shown in our experiments. The same argument holds for polynomial kernels with an odd exponent also. The flow of the reasoning for a polynomial kernel is similar to the Gaussian one, and we omit it.

Lemma 2 *The weighting scheme performed by the LFM-SVM algorithm increases the margin of the solution provided in input by the SVM.*

Proof: Consider the Gaussian radial basis function kernel (which we use in our experiments):

$$K(\mathbf{s}_i, \mathbf{x}) = e^{-\gamma||\mathbf{s}_i - \mathbf{x}||^2}. \tag{6.8}$$

The expression of the decision boundary in equation (2.35) becomes

$$f(\mathbf{x}) = \sum_{\mathbf{s}_i \in SV} \alpha_i y_i e^{-\gamma ||\mathbf{s}_i - \mathbf{x}||^2} - b = 0. \tag{6.9}$$

Consider now the j component of the gradient vector $\mathbf{n_d} = \nabla_{\mathbf{d}} f = (\frac{\partial}{\partial x_1} f_{\mathbf{d}}, \ldots, \frac{\partial}{\partial x_n} f_{\mathbf{d}})$ computed with respect to $\mathbf{x}$ at point $\mathbf{d}$:

$$n_{\mathbf{d},j} = \frac{\partial}{\partial x_j} f_{\mathbf{d}} = 2\gamma \sum_{\mathbf{s}_i \in SV} \alpha_i y_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2} \tag{6.10}$$

where $\mathbf{d}$ is the closest point to the query $\mathbf{x_0}$ on the boundary. Our local measure of relevance (6.1) for feature j is then given by

$$R_j(\mathbf{x}_0) = |n_{\mathbf{d},j}| = |2\gamma \sum_{\mathbf{s}_i \in SV} \alpha_i y_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2}|. \tag{6.11}$$

We partition SV in SV^- and SV^+, i.e. $SV = SV^- \cup SV^+$, where SV^- is the set of support vectors with label $y = -1$, and SV^+ is the set of support vectors with label $y = +1$. We can rewrite equation (6.11) as follows:

$$R_j(\mathbf{x}_0) = |2\gamma(\sum_{\mathbf{s}_i \in SV^+} \alpha_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2} - \sum_{\mathbf{s}_i \in SV^-} \alpha_i (s_{ij} - d_j) e^{-\gamma ||\mathbf{s}_i - \mathbf{d}||^2})|. \tag{6.12}$$

We want to identify the conditions that make $R_j(\mathbf{x}_0)$ large. We observe that γ, α_i, and the exponentials in (6.12) are all positive terms. A large value for $R_j(\mathbf{x}_0)$ is obtained when the terms $(s_{ij} - d_j)$ are all positive for $\mathbf{s}_i \in SV^+$ and all negative for $\mathbf{s}_i \in SV^-$, or vice versa. These conditions are satisfied when

$$d_j < s_{ij} \;\; \forall \mathbf{s}_i \in SV^+ \quad and \quad d_j > s_{ij} \;\; \forall \mathbf{s}_i \in SV^- \tag{6.13}$$

or

$$d_j > s_{ij} \;\; \forall \mathbf{s}_i \in SV^+ \quad and \quad d_j < s_{ij} \;\; \forall \mathbf{s}_i \in SV^-. \tag{6.14}$$

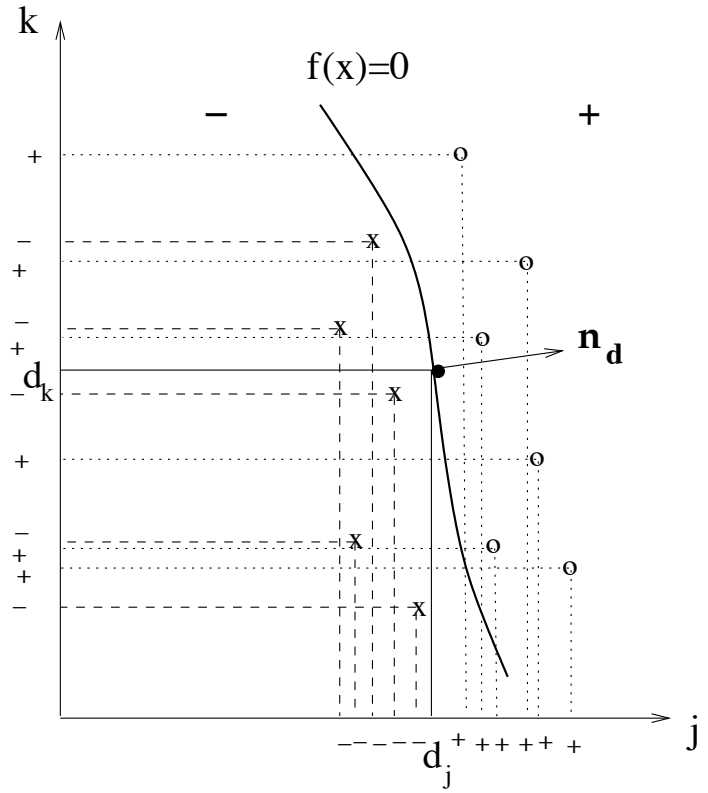

Figure 6.3: Illustration of a case in which conditions (6.13) are satisfied. Negative support vectors are represented as "x" and positive support vectors are denoted with "o". Along feature j, d_j separates the positive support vectors from the negative ones. Along feature k support vectors with opposite sign shuffle. A large weight is assigned to feature j and a small weight to feature k.

Figure 6.4 illustrates a case in which conditions (6.13) are satisfied. Along the orientation identified by feature j, d_j separates the positive support vectors from the negative ones. As a consequence, the value of $R_j(\mathbf{x}_0)$ is large, and a large weight is assigned to feature j. On the other hand, along the orientation identified by feature k, the negative support vectors mix with the positive support vectors. Therefore, the terms $(s_{ik} - d_k)$ becomes positive and negative for different $\mathbf{s}_i$ within either SV^+ or SV^-, and cancel each other in equation (6.12). As a result, the value of $R_k(\mathbf{x}_0)$ is small, and a small weight is assigned to feature k.

By assigning a large weight to feature j in Figure 6.4, and in general to input features close to the gradient direction, locally in neighborhood of support vectors, we increase the distance between the support vectors in SV^- and SV^+. This corresponds to improve the separability of classes along those orientations, and therefore the margin. As a consequence, better classification results can be achieved as also demonstrated in our experiments. ■

Chapter 7

LFM-SVM: Experimental Evaluation

This chapter presents an extensive experimental evaluation comparing LFM-SVM with a variety of classification approaches, including SVMs and ADAMENN.

7.1 Methods Compared

In the following we compare several classification methods using both simulated and real data. We compare the following classification approaches:

- LFM-SVM algorithm described in Figure 6.2. SVM^{light} [Joa99] with radial basis kernels is used to build the SVM classifier.
- RBF-SVM classifier with radial basis kernels. We used SVM^{light} [Joa99], and set the value of γ in $K(\mathbf{x}_i, \mathbf{x}) = e^{-\gamma||\mathbf{x}_i - \mathbf{x}||^2}$ equal to the optimal one determined via cross-validation. Also the value of C for the soft-margin classifier is optimized via cross-validation. The output of this classifier is the input of LFM-SVM.

- ADAMENN-adaptive metric nearest neighbor technique described in Chapter 4.
- Machete [Fri94] (see Chapter 3).
- Scythe [Fri94] (see Chapter 3).
- DANN-discriminant adaptive nearest neighbor classification [HT96a] (see Chapter 3).
- Simple K-NN method using the Euclidean distance measure.
- C4.5 decision tree method [Qui93].

In all the experiments, the features are first normalized over the training data to have zero mean and unit variance, and the test data features are normalized using the corresponding training mean and variance. Procedural parameters for each method were determined empirically through cross-validation over the training data, as described in Section 5.1.1. Each problem involves two classes.

7.2 Experiments on Simulated Data

For all simulated data, 10 independent training samples of size 200 were generated. For each of these, an additional independent test sample consisting of 200 observations was generated. These test data were classified by each competing method using the respective training data set. Error rates computed over all 2,000 such classifications are reported in Table 7.1.

7.2.1 The Problems

Multi-Gaussians. The data set consists of $q = 2$ input features, $N = 200$ training data, and $J = 2$ classes. Each class contains two spherical bivariate normal subclasses, having standard deviation 1. The mean vectors for one class are $(-3/4, -3)$ and $(3/4, 3)$; whereas for the other class are $(3, -3)$ and $(-3, 3)$. For each class, data are evenly drawn from each of the two normal subclasses. The first column of Table 7.1 shows the results for this problem. The standard deviations are: $0.17, 0.01$, $0.01, 0.01, 0.01$ $0.01, 0.01$ and 1.50, respectively.

Noisy-Gaussians. The data set consists of $q = 6$ input features, $N = 200$ training data, and $J = 2$ classes. The data for this problem are generated as in the previous example, but augmented with four predictors having independent standard Gaussian distributions. They serve as noise. For each class, data are evenly drawn from each of the two normal subclasses. Results are shown in the second column of Table 7.1. The standard deviations are: 0.18, 0.01, 0.02, 0.01, 0.01, 0.01, 0.01 and 1.60, respectively.

7.2.2 Results

Table 7.1 shows that all methods have similar performances for the MultiGaussians problem, with C4.5 being the worst performer. When the noisy predictors are added to the problem (NoisyGaussians), we observe different levels of deterioration in performance among the eight methods. LFM-SVM shows the most robust behavior in presence of noise. K-NN is instead the worst performer. We also observe that C4.5 has similar error rates in both cases; we noticed, in fact, that for the majority of the 10 independent trials we run it uses only the first two input features to

build the decision tree. In Figure 7.2.2 we plot the performances of LFM-SVM and RBF-SVM as a function of an increasing number of noisy features (for the same MultiGaussians problem). The standard deviations for RBF-SVM (in order of increasing number of noisy features) are: $0.01, 0.01, 0.03, 0.03, 0.03$ and 0.03. The standard deviations for LFM-SVM are: $0.17, 0.18, 0.2, 0.3, 0.3$ and 0.3. The LFM-SVM technique shows a considerable improvement over RBF-SVM as the amount of noise increases.

Table 7.1: Average classification error rates for simulated data.

	MultiGauss	NoisyGauss
LFM-SVM	**3.3**	**3.4**
RBF-SVM	**3.3**	5.3
ADAMENN	3.4	4.1
Machete	3.4	4.3
Scythe	3.4	4.8
DANN	3.7	4.7
K-NN	**3.3**	7.0
C4.5	5.0	5.1

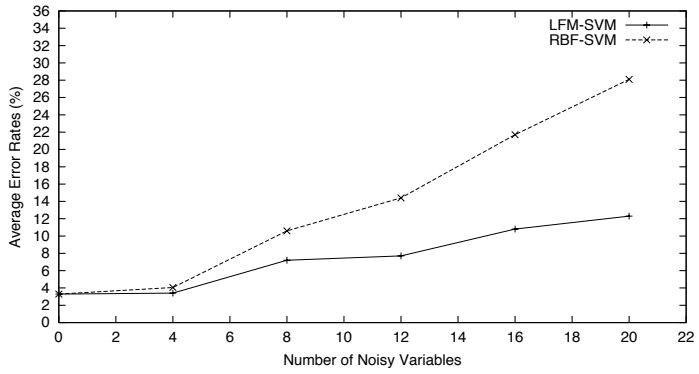

Figure 7.1: Average error rates of LFM-SVM and RBF-SVM as a function of an increasing number of noisy predictors.

7.3 Experiments on Real Data

In our experiments we used seven different real data sets. They are all taken from UCI Machine Learning Repository at http://www.cs.uci.edu/∼mlearn/MLRepository.html. If an attribute of a data set has missing values, we eliminate that feature from the entire set. For the Iris, Sonar, Liver and Vote data we perform leave-one-out cross-validation to measure performance, since the number of available data is limited for these data sets. For the Breast, OQ-letter and Pima data we randomly generated five independent training sets of size 200. For each of these, an additional independent test sample consisting of 200 observations was generated. Table 7.2 shows the cross-validated error rates for the eight methods under consideration on the seven real data.

7.3.1 The Problems

1. **Iris data**. See Section 5.3.1 for a description. The results on this data set are shown in the first column of Table 7.2.

2. **Sonar data**. See Section 5.3.1 for a description. The results on this data set are shown in the second column of Table 7.2.

3. **Liver data**. See Section 5.3.1 for a description. The results on this data set are shown in the third column of Table 7.2.

4. **Vote data**. This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The data set consists of $N = 232$ instances after removing missing values, and $J = 2$ classes (democrat and republican). The instances are represented by $q = 16$ boolean valued features. The average leave-one-out cross-validation error rates

are shown in the fourth column of Table 7.2.

5. **Wisconsin breast cancer data**. The data set consists of $q = 9$ medical input features that are used to make a binary decision on the medical condition: determining whether the cancer is malignant or benign. The data set contains 683 examples after removing missing values. Average error rates for this problem are shown in the fifth column of Table 7.2. The standard deviations are: $0.2, 0.2, 0.2, 0.2, 0.2, 0.9, 0.9$ and 0.9, respectively.

6. **OQ data**. This data set consists of $q = 16$ numerical attributes and $J = 2$ classes. The objective is to identify black-and-white rectangular pixel displays as one of the two capital letters "O" and "Q" in the English alphabet. There are $N = 1536$ instances in this data set. The character images were based on 20 different fonts, and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. The average error rates over five independent runs are shown in the sixth column of Table 7.2. The standard deviations are: 0.2, 0.2, 0.2, 0.3, 0.2, 1.1, 1.5 and 2.1, respectively.

7. **Pima Indians Diabete data**. This data set consists of $q = 8$ numerical medical attributes and $J = 2$ classes (tested positive or negative for diabetes). There are $N = 768$ instances. Average error rates over five independent runs are shown in the seventh column of Table 7.2. The standard deviations are: $0.4, 0.4, 0.4, 0.4, 0.4, 2.4, 2.1$ and 0.7, respectively.

Table 7.2: Average classification error rates for real data

	Iris	Sonar	Liver	Vote	Breast	OQ	Pima
LFM-SVM	4.0	11.0	28.1	**2.6**	3.0	3.5	**19.3**
RBF-SVM	4.0	12.0	**26.1**	3.0	3.1	3.4	21.3
ADAMENN	**3.0**	9.1	30.7	3.0	3.2	**3.1**	20.4
Machete	5.0	21.2	27.5	3.4	3.5	7.4	20.4
Scythe	4.0	16.3	27.5	3.4	2.7	5.0	20.0
DANN	6.0	**7.7**	30.1	3.0	**2.2**	4.0	22.2
K-NN	6.0	12.5	32.5	7.8	2.7	5.4	24.2
C4.5	8.0	23.1	38.3	3.4	4.1	9.2	23.8

7.3.2 Results

Table 7.2 shows that LFM-SVM achieves the best performance in 2/7 of the real data sets; in one case it shows the second best performance, and in the remaining four its error rate is still quite close to the best one.

It seems natural to quantify this notion of robustness; that is, how well a particular method m performs on average across the problems taken into consideration. Following Friedman [Fri94], we capture robustness by computing the ratio b_m of the error rate e_m of method m and the smallest error rate over all methods being compared in a particular example:

$$b_m = e_m / \min_{1 \leq k \leq 8} e_k.$$

Figure 7.2 plots the distribution of b_m for each method over the seven real data sets. The dark area represents the lower and upper quartiles of the distribution that are separated by the median. The outer vertical lines show the entire range of values for the distribution. The outer vertical lines for the LFM-SVM method are not visible because they coincide with the limits of the

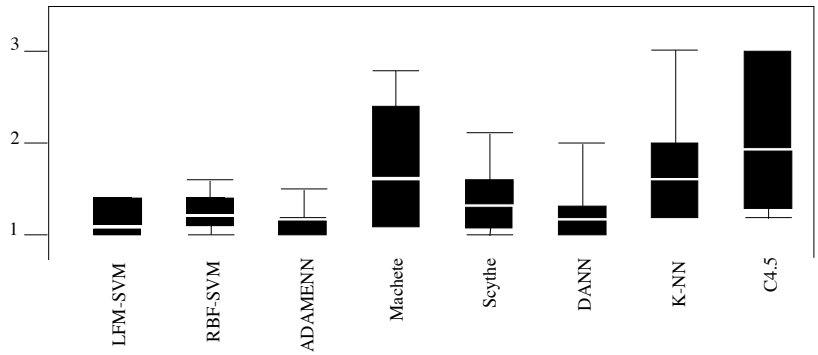

Figure 7.2: Performance distributions for real data.

lower and upper quartiles. The spread of the error distribution for LFM-SVM is narrow and close to one. The spread for ADAMENN has a similar behavior. The results clearly demonstrate that LFM-SVM and ADAMENN obtained the most robust performance over the data sets.

The poor performance of the machete and C4.5 methods might be due to the greedy strategy they employ. Such recursive peeling strategy removes at each step a subset of data points permanently from further consideration. As a result, changes in an early split, due to any variability in parameter estimates, can have a significant impact on later splits, thereby producing different terminal regions. This makes predictions highly sensitive to the sampling fluctuations associated with the random nature of the process that produces the traning data, thus leading to high variance predictions. The scythe algorithm, by relaxing the winner-take-all splitting strategy of the machete algorithm, mitigates the greedy nature of the approach, and thereby achieves better performance.

In [HT96a], the authors show that the metric employed by the DANN algorithm approximates the weighted Chi-squared distance, given that class densities are Gaussian and have the same covariance matrix. As a consequence, we may expect a degradation in performance when the data do not follow Gaussian distributions and are corrupted by noise, which is likely the case in real

scenarios like the ones tested here.

We observe that the sparse solution given by SVMs provides LFM-SVM with principled guidelines to efficiently set the input parameters. This is an important advantage over ADAMENN, which has six tunable input parameters. Furthermore, LFM-SVM speeds up the classification process since it applies the nearest neighbor rule only once, whereas ADAMENN applies it at each point within a region centered at the query. We also observe that the construction of the SVM for LFM-SVM is carried out off-line only once, and there exist algorithmic and computational results which make SVM training practical also for large-scale problems [Joa99, Pla99, CP00], and extend the SVM learning algorithm in an incremental fashion [SLS99, MMP00, DG01a]. The major limitation of the LFM-SVM algorithm is that, like support vector machines, can be applied to classification problems with two classes only. On the other hand, ADAMENN, like nearest neighbor methods, can be applied to problems with an arbitrary number of classes.

The LFM-SVM offers performance improvements over the RBF-SVM algorithm alone, for both the (noisy) simulated and real data sets. The reason for such performance gain may rely on the effect of our local weighting scheme on the separability of classes, and therefore on the margin, as shown in Section 6.4. Assigning large weights to input features close to the gradient direction, locally in neighborhoods of support vectors, corresponds to increase the spatial resolution along those orientations, and therefore to improve the separability of classes. As a consequence, better classification results can be achieved as demonstrated in our experiments.

7.4 Discussion

We have described a locally adaptive metric classification method and demonstrated its efficacy through experimental results. The proposed technique offers performance improvements over the SVM alone, and has the potential of scaling up to large data sets. It speeds up, in fact, the classification process by computing off-line the information relevant to define local weights. It also applies the nearest neighbor rule only once, whereas ADAMENN applies it at each point within a region centered at the query. LFM-SVM benefits from the sparse solution given by SVMs, and efficiently set the input parameters according to principled guidelines. The major advantage of ADAMENN over LFM-SVM is that is well suited for problems with an arbitrary numbers of classes.

A considerable amount of work has been done for efficient discovery of nearest neighbors. Unfortunately, all methods suffer from the curse of dimensionality, and therefore they become less effective in high dimensions. Our scheme could also be used to improve the effectiveness of nearest neighbor search in high dimensions by virtue of the local dimensionality reduction resulting from the feature weights. We intend to further explore this direction of research in our future work.

Chapter 8

Within-Cluster Adaptive Metric for Clustering

In this chapter, we introduce an algorithm that discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques. Our method associates to each cluster a weight vector, whose values give information of the degree of relevance of features for each set in the partition. We formally prove that our algorithm converges, and experimentally demonstrate (Chapter 9) the gain in perfomance we achieve with our method.

8.1 Introduction

The clustering problem concerns the discovery of homogeneous groups of data according to a certain similarity measure. It has been studied extensively in statistics [AH96], machine learning

[MS83, CS96], and database communities [NH94, EKX95, ZRL96].

Given a set of multi-dimensional data, (partitional) clustering finds a partition of the points into clusters such that the points within a cluster are more similar to each other than to points in different clusters. The popular K-means or K-medoids methods compute one representative point per cluster, and assign each object to the cluster with the closest representative, so that the sum of the squared differences between the objects and their representatives is minimized. Finding a set of representative vectors for clouds of multi-dimensional data is an important issue in data compression [GG91], signal coding [Ger82, GG91], pattern classification [DH73], and function approximation tasks [MD89, PG90].

Clustering suffers from the curse of dimensionality problem in high dimensional spaces. In high dimensional spaces, it is highly likely that, for any given pair of points within the same cluster, there exist at least a few dimensions on which the points are far apart from each other. It is not meaningful to look for clusters in such a high dimensional space as the average density of points anywhere in input space is likely to be low. As a consequence, distance functions that equally use all input features may be ineffective.

Furthermore, several clusters may exist in different subspaces, comprised of different combinations of features. In many real world problems, in fact, some points are correlated with respect to a given set of dimensions, and others are correlated with respect to different dimensions. Each dimension could be relevant to at least one of the clusters.

The problem of high dimensionality could be addressed by requiring the user to specify a subspace (i.e., subset of dimensions) for cluster analysis. However, the identification of subspaces

by the user is an error-prone process. More importantly, correlations that identify clusters in the data are likely not to be known by the user. Indeed, we desire such correlations, and induced subspaces, to be part of the findings of the clustering process itself.

An alternative solution to high dimensional settings consists in reducing the dimensionality of the input space. Traditional feature selection algorithms select certain dimensions in advance. Methods such as PCA [DH73, Fuk90] transform the original input space into a lower dimensional space by constructing dimensions that are linear combinations of the given features, and are ordered by nonincreasing variance. While PCA may succeed in reducing the dimensionality, it has major drawbacks. The new dimensions can be difficult to interpret, making it hard to understand clusters in relation to the original space. Furthermore, all global dimensionality reduction techniques (like PCA) are not effective in identifying clusters that may exist in different subspaces. In this situation, in fact, since data across clusters manifest different correlations with features, it may not always be feasible to prune off too many dimensions without incurring a loss of crucial information. This is because each dimension could be relevant to at least one of the clusters.

These limitations of global dimensionality reduction techniques suggest that, to capture the local correlations of data, a proper feature selection procedure should operate locally in input space. Local feature selection allows to embed different distance measures in different regions of the input space; such distance metrics reflect local correlations of data. In this paper we propose a *soft* feature selection procedure that assigns (local) weights to features according to the local correlations of data along each dimension. Dimensions along which data are loosely correlated receive a small weight, that has the effect of elongating distances along that dimension. Features along which data are strongly correlated receive a large weight, that has the effect of constricting

distances along that dimension. Figure 8.1 gives a simple example. The left plot depicts two clusters of data elongated along the x and y dimensions. The right plot shows the same clusters, where within-cluster distances between points are computed using the respective local weights generated by our algorithm (GenProClus). The weight values reflect local correlations of data, and reshape each cluster as a *dense spherical cloud*. This directional local reshaping of distances better separates clusters, and allows for the discover of different patterns in different subspaces of the original input space.

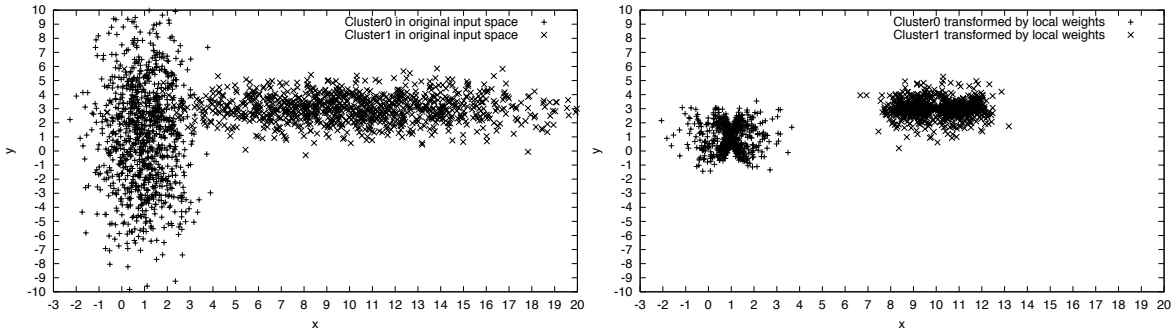

Figure 8.1: (Left) Clusters in original input space. (Right) Clusters transformed by local weights.

8.2 Related Work

Local dimensionality reduction approaches for the purpose of efficiently indexing high dimensional spaces have been recently discussed in the database literature [TCL98, CM00, KCM+01]. Applying global dimensionality reduction techniques when data are not globally correlated can cause significant loss of distance information, resulting in a large number of false positives and hence a high query cost. The general approach adopted by the authors is to find local correlations in the data, and perform dimensionality reduction on the locally correlated clusters individually. For

example, in [CM00], the authors first construct spatial clusters in the original input space using a simple tecnique that resembles K-means. Principal component analysis is then performed on each spatial cluster individually to obtain the principal components.

In general, the efficacy of these methods depends on how the clustering problem is addressed in the first place in the original feature space. A potential serious problem with such techniques is the lack of data to locally perform PCA on each cluster to derive the principal components. Moreover, for clustering purposes, the new dimensions may be difficult to interpret, making it hard to understand clusters in relation to the original space.

The problem of finding different clusters in different subspaces of the original input space has been addressed in [AGG+98]. The authors use a density based approach to identify clusters. The algorithm (CLIQUE) proceeds from lower to higher dimensionality subspaces and discovers dense regions in each subspace. To approximate the density of the points, the input space is partitioned into cells by dividing each dimension into the same number ξ of equal length intervals. For a given set of dimensions, the cross product of the corresponding intervals (one for each dimension in the set) is called a *unit* in the respective subspace. A unit is dense if the number of points it contains is above a given threshold τ. Both ξ and τ are parameters defined by the user. The algorithm finds all dense units in each k-dimensional subspace by building from the dense units of $(k-1)$-dimensional subspaces, and then connects them to describe the clusters as union of maximal rectangles.

While the work in [AGG+98] successfully introduces a methodology for looking at different subspaces for different clusters, it does not compute a partitioning of the data into disjoint groups. The reported dense regions largely overlap, since for a given dense region all its projections

on lower dimensionality subspaces are also dense, and they all get reported. On the other hand, for many applications such as customer segmentation and trend analysis, a partition of the data is desirable since it provides a clear interpretability of the results.

[DB00] also addresses the problem of feature selection to find clusters hidden in high dimensional data. The authors search through feature subset space, evaluating each subset by first clustering in the corresponding subspace, and then evaluating the resulting clusters and feature subset using the chosen feature selection criterion. The two feature selection criteria investigated are the scatter separability used in discriminant analysis [Fuk90], and a maximum likelihood criterion. A sequential forward greedy strategy [Fuk90] is employed to search through possible feature subsets. We observe that dimensionality reduction is performed globally in this case. Therefore, the technique in [DB00] is expected to be effective when a data set contains some relevant features and some irrelevant (noisy) ones, across all clusters.

The problem of finding different clusters in different subspaces is also addressed in [APW+99]. The proposed algorithm (PROjected CLUStering) seeks subsets of dimensions such that the points are closely clustered in the corresponding spanned subspaces. Both the number of clusters and the average number of dimensions per cluster are user-defined parameters. PROCLUS starts with choosing a random set of medoids, and then progressively improves the quality of medoids by performing an iterative hill climbing procedure that discards the 'bad' medoids from the current set. In order to find the set of dimensions that matter the most for each cluster, the algorithm selects the dimensions along which the points have the smallest average distance from the current medoid. The authors do not prove that the algorithm converges to the optimality criterion they choose.

Our method (that we call GENeralized PROjected CLUStering) can be seen as a generalization of PROCLUS. Our method does not require to specify the average number of dimensions to be kept per cluster. For each cluster, in fact, *all* features are taken into consideration, but properly weighted. The PROCLUS algorithm is more prone to loss of information if the number of dimensions is not properly chosen. For example, if data of two clusters in two dimensions are distributed as in Figure 8.2, PROCLUS may find that feature x is the most important for cluster 0, and feature y is the most important for cluster 1. But projecting cluster 1 along the y dimension doesn't allow to properly separate points of the two clusters. We avoid this problem by keeping both dimensions for both clusters, and properly weighting distances along each feature within each cluster.

Generative approaches have also been developed for local dimensionality reduction and clustering. The approach in [GH96] makes use of maximum likelihood factor analysis to model local correlations between features. The resulting generative model obeys the distribution of a mixture of factor analyzers. An expectation-maximization algorithm is presented for fitting the parameters of the mixture of factor analyzers. The choice of the number of factor analyzers, and the number of factors in each analyzer (that drives the dimensionality reduction) remain an important open issue for the approach in [GH96].

[TB99] extends the single PCA model to a mixture of local linear sub-models to capture nonlinear structure in the data. A mixture of principal component analyzers model is derived as a solution to a maximum-likelihood problem. An EM algorithm is formulated to estimate the parameters.

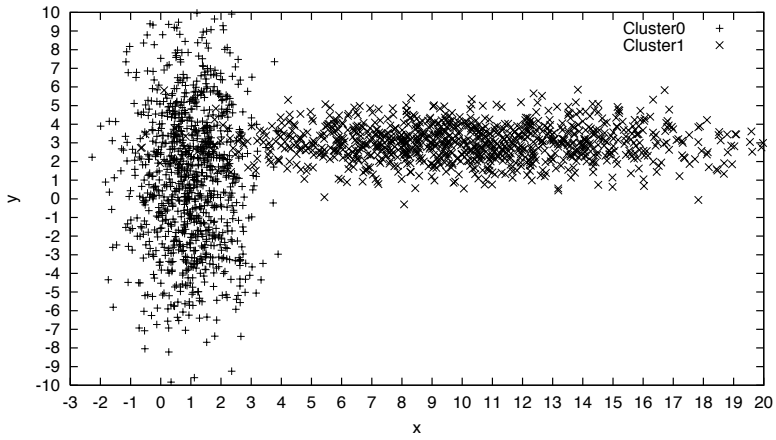

Figure 8.2: Distributions of two clusters in two dimensions.

While the methods in [GH96, TB99], as well as the standard mixture of Gaussians technique, are generative and parametric, GenProClus can be seen as an attempt to directly estimate from the data local correlations between features. Furthermore, both mixture models in [GH96, TB99] inherit the soft clustering component of the EM update equations. On the contrary, GenProClus computes a partitioning of the data into disjoint groups. As previously mentioned, for many data mining applications a partition of the data is desirable since it provides a clear interpretability of the results. We finally observe that, while mixture of Gaussians models, with arbitrary covariance matrices, could in principle capture local correlations along any directions, lack of data to locally estimate full covariance matrices in high dimensional spaces is a serious problem in practice.

8.3 Problem Statement

We define what we call *weighted cluster*. Consider a set of points in some space of dimensionality q. A *weighted cluster* G is a subset of data points, together with a vector of weights

$\mathbf{w} = (w_1, \ldots, w_q)$, such that the points in G are closely clustered according to the L_2 norm distance weighted using $\mathbf{w}$. The component w_j measures the degree of correlation of points in G along feature j. The problem becomes now how to estimate the weight vector $\mathbf{w}$ for each cluster in the data set.

In this setting, the concept of *cluster* is not based only on points, but also involves a weighted distance metric, i.e., clusters are been discovered in spaces transformed by $\mathbf{w}$. Each cluster is associated with its own $\mathbf{w}$, that reflects the correlation of points in the cluster itself. The effect of $\mathbf{w}$ is to transform distances so that the associated cluster is reshaped into a dense hypersphere of points separated from other data.

In traditional clustering, the partition of a set of points is induced by a set of *representative* vectors, also called *centroids* or *centers*. The partition induced by discovering weighted clusters is formally defined as follows.

Definition Given a set S of N points $\mathbf{x}$ in q-dimensional Euclidean space, a set of k centers $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}, \mathbf{c}_j \in \Re^q, j = 1, \ldots, k$, coupled with a set of corresponding weight vectors $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$, $\mathbf{w}_j \in \Re^q, j = 1, \ldots, k$, partition S into k sets $\{S_1, \ldots, S_k\}$:

$$S_j = \{\mathbf{x} | (\sum_{i=1}^{q} w_{ji}(x_i - c_{ji})^2)^{1/2} < (\sum_{i=1}^{q} w_{li}(x_i - c_{li})^2)^{1/2}, l \neq j\} \tag{8.1}$$

The set of centers and weights is *optimal* with respect to the Euclidean norm, if they minimize the error measure:

$$E_1(C,W) = \sum_{j=1}^{k}\sum_{i=1}^{q} w_{ji} e^{(X_j - X_{ji})} \tag{8.2}$$

subject to the constraints $\sum_{i=1}^{q} w_{ji}^2 = 1 \; \forall j$. C and W are $(q \times k)$ matrices whose column vectors are $\mathbf{c}_j$ and $\mathbf{w}_j$ respectively, i.e. $C = [\mathbf{c}_1 \ldots \mathbf{c}_k]$ and $W = [\mathbf{w}_1 \ldots \mathbf{w}_k]$. X_j and X_{ji} are defined as follows:

$$X_j = \frac{1}{|S_j|}(\max_l(\sum_{\mathbf{x}\in S_j}(c_{jl} - x_l)^2)) \tag{8.3}$$

$$X_{ji} = \frac{1}{|S_j|}(\sum_{\mathbf{x}\in S_j}(c_{ji} - x_i)^2) \tag{8.4}$$

where $|S_j|$ is the cardinality of set S_j. X_{ji} represents the average distance from the centroid $\mathbf{c}_j$ of points in cluster j along dimension i, and X_j is the largest of such average distances $\forall i$.

The minimization of the error function E_1 as defined in (8.2) has a specific geometric interpretation. The optimal solution aims to minimize, for each cluster, the (exponential of the) discrepancy between the largest spread (X_j) of data along each dimension, and each of such spreads (X_{ji}). As a result, in the space transformed by optimal weights, the corresponding cluster has the shape of a hypersphere well separated from other data. The exponential function in (8.2) has the effect of making the weights w_{ji} more sensitive to the discrepancy $(X_j - X_{ji})$, and therefore to changes in local feature relevance. As a consequence, clusters are better separated in the transformed spaces, and large performance improvements can be achieved as also demonstrated with our experimental results.

In the following we present an algorithm that finds a solution (set of centers and weights) that is a local minimum of the error function (8.2).

8.4 Generalized Projected Clustering Algorithm

We start with *well-scattered* points in S as the k centroids: we choose the first centroid at random, and select the others so that they are far from one another, and from the first chosen center. We initially set the values of weights to 1. We progressively improve the quality of the centroids and of the weights by investigating the space near the centers, in order to estimate the dimensions that matter the most, i.e. the dimensions along which local data are mostly correlated. Specifically, we proceed as follows.

Given the initial centroids $\mathbf{c}_j$, for $j = 1, \ldots, k$, we compute the corresponding sets S_j as defined in (8.1), where $w_{ji} = 1\ \forall j$ and $\forall i$. We then compute the average distance along each dimension from the points in S_j to $\mathbf{c}_j$. Let X_{ji} denote this average distance along dimension i, and let X_j the largest average distance among the N dimensions for cluster j. The smaller X_{ji} is, the larger is the correlation of points along dimension i. Thus, the difference $X_j - X_{ji}$ gives us a value that is proportional to the amount of correlation of points along feature i. Let $X'_{ji} = X_j - X_{ji}$. We use the value X'_{ji} in an exponential weighting scheme to credit weights to features (and to clusters):

$$w_{ji} = exp(h \times X'_{ji}) / (\sum_{l=1}^{q} (exp(h \times 2 \times X'_{jl})))^{1/2} \tag{8.5}$$

where h is a parameter that can be chosen to maximize (minimize) the influence of X'_{ji} on w_{ji}[1]. When $h = 0$ we have $w_{ji} = 1/q$, thereby ignoring any difference between the X'_{ji}. On the other hand, when h is large a change in X'_{ji} will be exponentially reflected in w_{ji}. We empirically determine the value of h through cross-validation in our experiments. The exponential weighting is more sensitive to changes in local feature relevance [BV92] and gives rise to better performance

[1] h is the analogous of the c parameter in equation 4.8, and of the A parameter in equation 6.3

improvement. In fact, it is more stable because it prevents distances from extending infinitely in any direction, i.e., zero weight. This, however, can occur when either linear or quadratic weighting is used.

These weights w_{ji} enable to elongate distances along less important dimensions, i.e. dimensions along which points are loosely correlated, and, at the same time, to constrict distances along the most influential ones, i.e. features along which points are strongly correlated. Note that the technique is centroid-based because weightings depend on the centroid.

The computed weights are used in equation (8.1) to update the sets S_j. Then, the new partition is used to recompute the centroid coordinates:

$$\mathbf{c}_j = \frac{\sum_{\mathbf{x}} \mathbf{x} 1_{S_j}(\mathbf{x})}{\sum_{\mathbf{x}} 1_{S_j}(\mathbf{x})}$$

for each $j = 1, \ldots, k$, where $1_S(.)$ is the indicator function of set S. The procedure is iterated until convergence is reached, that is when no change in the coordinates of centers is observed. The resulting algorithm, that we call GenProClus, is summarized in Figure 8.3.

8.5 Convergence of the GenProClus Algorithm

To formally prove convergence of the GenProClus algorithm we need an error function that is differentiable with respect to both w_{ji} and c_{ji}. We observe that the error measure in (8.2), while sound in theory, is not differentiable due to the definition of X_j in terms of a max function. We solve this problem by substituting X_j with a value X that measures the largest spread of the projections of data in any dimensions. We observe that X is constant given the data set, and therefore does not depend on w_{ji} or c_{ji}. X provides an upper bound for all X_j, $j = 1, \ldots, k$. Thus, the

Input: N points $\mathbf{x} \in R^q$, and the number of clusters k.

1. Start with k initial centroids $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$;
2. Set $w_{ji} = 1$, for each centroid $\mathbf{c}_j, j = 1, \ldots, k$ and each feature $i = 1, \ldots, q$;
3. For each centroid $\mathbf{c}_j, j = 1, \ldots, k$, and for each data point $\mathbf{x}$:
   - Set $S_j = \{\mathbf{x} | j = arg\min_l WDist(\mathbf{c}_l, \mathbf{x})\}$,
     where $WDist(\mathbf{c}_l, \mathbf{x}) = (\sum_{i=1}^{q} w_{li}(c_{li} - x_i)^2)^{1/2}$;
4. **Compute new weights**:
   For each centroid $\mathbf{c}_j, j = 1, \ldots, k$, and for each feature i:
   - Set $X_{ji} = \sum_{\mathbf{x} \in S_j} (c_{ji} - x_i)^2 / |S_j|$, where $|S_j|$ is the cardinality of set S_j;
   - Set $X_j = \max_i X_{ji}$;
   - Set $X'_{ji} = X_j - X_{ji}$;
   - Set $w_{ji} = exp(h \times X'_{ji}) / \sum_{l=1}^{N} (exp(h \times 2 \times X'_{jl}))^{1/2}$;
5. For each centroid $\mathbf{c}_j, j = 1, \ldots, k$, and for each data point $\mathbf{x}$:
   - Recompute $S_j = \{\mathbf{x} | j = arg\min_l WDist(\mathbf{c}_l, \mathbf{x})\}$;
6. **Compute new centroids**:
   Set $\mathbf{c}_j = \frac{\sum_{\mathbf{x}} \mathbf{x} 1_{S_j}(\mathbf{x})}{\sum_{\mathbf{x}} 1_{S_j}(\mathbf{x})}$, for each $j = 1, \ldots, k$, where $1_S(.)$ is the indicator function of set S;
7. Iterate 3, 4, 5, and 6 until convergence (i.e., no change in centroid coordinates)

Output: Set of centroid and weight vectors $\mathbf{c}_j, \mathbf{w}_j$, for $j = 1, \ldots, k$.

Figure 8.3: The GenProClus algorithm

resulting error function obeys the same principle that motivates the original error measure (8.2).

The resulting error function to be considered is:

$$E_2(C,W) = \sum_{j=1}^{k}\sum_{i=1}^{q} w_{ji} e^{(X - X_{ji})} \tag{8.6}$$

where $X = \frac{1}{N}(\max_l(\sum_{\mathbf{x}\in S}(m_l - x_l)^2))$, $\mathbf{m} = \frac{1}{N}\sum_{\mathbf{x}\in S}\mathbf{x}$, and X_{ji} is defined in equation (8.4). Our objective becomes the minimization of (8.6) subject to the constraints $\sum_i {w_{ji}}^2 = 1\ \forall j$. We can solve this constrained optimization problem by introducing the Lagrange multipliers λ_j (one for each constraint), and minimizing the resulting (unconstrained now) error function

$$E(C,W) = \sum_{j=1}^{k}\sum_{i=1}^{q} w_{ji} e^{(X - X_{ji})} + \sum_{j=1}^{k} \lambda_j (1 - \sum_{i=1}^{q} w_{ji}^2) \tag{8.7}$$

We prove the following theorem.

Theorem 1 *The GenProClus algorithm converges to a local minimum of the error function (8.7).*

Proof: For a fixed partition P and fixed c_{ji}, we compute the optimal w_{ji} by setting $\frac{\partial E}{\partial w_{ji}} = 0$ and $\frac{\partial E}{\partial \lambda_j} = 0$. We obtain:

$$\frac{\partial E}{\partial w_{ji}} = e^{X - X_{ji}} - 2\lambda_j w_{ji} = 0 \tag{8.8}$$

$$\frac{\partial E}{\partial \lambda_j} = 1 - \sum_{i=1}^{q} w_{ji}^2 = 0 \tag{8.9}$$

Solving equation (8.8) with respect to w_{ji} we obtain $w_{ji} = \frac{e^{X - X_{ji}}}{2\lambda_j}$. Substituting this expression in equation (8.9), and solving with respect to λ_j we obtain $\lambda_j = \frac{1}{2}(\sum_{i=1}^{q} e^{2(X - X_{ji})})^{1/2}$.

Thus, the optimal w'_{ji} is

$$w'_{ji} = \frac{e^{X-X_{ji}}}{(\sum_{l=1}^{q} e^{2(X-X_{jl})})^{1/2}} \tag{8.10}$$

as in Step 4 of the GenProClus algorithm.

For a fixed partition P and fixed w_{ji}, we compute the optimal c'_{ji} by setting $\frac{\partial E}{\partial c_{ji}} = 0$. We obtain:

$$\frac{\partial E}{\partial c_{ji}} = w_{ji} e^{(X-X_{ji})}(-\frac{2}{|S_j|}\sum_{\mathbf{x}\in S_j}(c_{ji} - x_i)) = 0 \tag{8.11}$$

Solving equation (8.11) with respect to c_{ji} we obtain:

$$c'_{ji} = \frac{1}{|S_j|}\sum_{\mathbf{x}\in S_j} x_i \tag{8.12}$$

as in Step 6 of the GenProClus algorithm.

The algorithm consists in repeatedly replacing w_{ji} and c_{ji} with w'_{ji} and c'_{ji} using equations (8.10) and (8.12), respectively. The value of the error function E at completion of each iteration is $E_{P_1}(C', W', P_1)$, where we explicit the dependence of E on the partition of points P_1 computed in Step 5 of the algorithm. C' and W' are the matrices of the newly computed centroids and weights. Since the new partition P' computed in Step 3 of the successive iteration is by definition the best assignment of points $\mathbf{x}$ to the centroids c'_{ji} according to the weighted Euclidean distance with weights w'_{ji}, we have the following inequality:

$$E_{P'}(C', W', P') - E_{P_1}(C', W', P_1) \leq 0 \tag{8.13}$$

Using this result, and the identities $E(C', W') = E_{P'}(C', W', P')$ and $E(C, W) =$

$E_P(C, W, P)$, we can derive the following inequality:

$$\begin{aligned} E(C', W') - E(C, W) &= E_{P'}(C', W', P') - E_{P_1}(C', W', P_1) + E_{P_1}(C', W', P_1) - E_P(C, W, P) \\ &\leq E_{S_1}(C', W', S_1) - E_S(C, W, S) \leq 0 \end{aligned} \tag{8.14}$$

where the last inequality (8.14) is derived by using the definitions of u'_{ji} and c'_{ji}.

Thus, each iteration of the algorithm decreases the lower bounded error function E (8.7) until the error reaches a fixed point where conditions $\mathbf{w}_j^{\star\prime} = \mathbf{w}_j^\star, \mathbf{c}_j^{\star\prime} = \mathbf{c}_j^\star \; \forall j$ are verified. The fixed points $\mathbf{w}_j^\star$ and $\mathbf{c}_j^\star$ give a local minimum of the error function E. ■

It is interesting to point out the analogy of this derivation with the mathematics of the *EM* algorithm [DLR77]. The hidden variables are the assignments of the points to the centroids. Step 3 constitutes the *E* step of the *EM* algorithm: it finds the values of the hidden variables S_j given the previous values of the parameters w_{ji} and c_{ji}. The following step (*M* step) consists in finding new matrices of weights and centroids that minimize E_P and E_{P_1} respectively.

Chapter 9

GenProClus: Experimental Evaluation

This chapter presents an experimental evaluation comparing GenProClus, EM for mixtures of Gaussians, and K-means algorithms, using a variety of simulated data sets.

9.1 Experimental Settings

In our experiments we have designed nine different simulated data sets. Clusters are distributed according to multivariate gaussians with different mean and standard deviation vectors. We have tested problems with two and three clusters up to 50 dimensions. For each problem, we have generated 10 training data sets, and for each of them an independent test set. In the following we report performance results obtained via 10-fold cross-validation comparing GenProClus, EM and K-means algorithms. The Matlab code written by David Corney available at http://www.cs.ucl.ac.uk/staff/D.Corney/ClusteringMatlab.html was used for the EM algorithm. The k centroids for the three algorithms are initialized by choosing well-scattered points among the

given data. EM estimates full covariance matrices. To facilitate the interpretation of weight values, we require that $\sum_i w_{ji} = 1 \; \forall j$ in our experiments, by properly adjusting the normalization factor of the weighting scheme (8.5).

9.1.1 The Problems

1. **Example1**. This data set consists of $q = 2$ attributes and $J = 2$ clusters. Data for one cluster are generated from a multivariate normal distribution with mean vector $(1, 1)$ and standard deviations $(1, 4)$. Data for the other cluster are generated from a normal distribution with mean vector $(10, 3)$ and standard deviations $(4, 1)$. Figure 9.1 shows the distributions of data for the two clusters. Average results obtained over 10 independent training and testing sets of size 2000 each are shown in Table 9.1.

2. **Example2**. This data set consists of $q = 3$ attributes and $J = 2$ clusters. One cluster is drawn from a multivariate normal distribution with mean vector $(1, 1, 1)$, and standard deviations $(1, 4, 1)$. The second cluster is drawn again form a multivariate normal distribution with mean vector $(5, 5, 1)$, and standard deviations $(1, 1, 4)$. Average results obtained over 10 independent training and testing sets of size 2000 each are shown in Table 9.1.

3. **Example3**. This data set consists of $q = 3$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, 1, 1)$ are $(1, 4, 1)$ respectively. For the other cluster the vectors are $(3, 1, 1)$ and $(1, 1, 4)$. Figures 9.1.1-9.3 shows the two clusters projected in $x - y$, $x - z$, and $y - z$ spaces respectively. Table 9.1 shows the results for this problem. We generated 40000 data points, and performed 10-fold cross-validation with 20000 training data and 20000 testing data.

4. **Example4**. This data set consists of $q = 5$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, 1, 1, 1, 1)$ and $(1, 4, 1, 4, 1)$, respectively. For the other cluster the vectors are $(5, 1, 1, 1, 1)$ and $(4, 1, 1, 1, 4)$. Table 9.1 shows the results for this problem. We generated 40000 data points, and performed 10-fold cross-validation with 20000 training data and 20000 testing data.

5. **Example5**. This data set consists of $q = 2$ input features and $J = 3$ clusters. All three clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(2, 0)$ are $(4, 1)$, respectively. For the second cluster the vectors are $(10, 0)$ and $(1, 4)$, and for the third are $(18, 0)$ and $(4, 1)$. Table 9.1 shows the results for this problem. We generated 60000 data points, and performed 10-fold cross-validation with 30000 training data and 30000 testing data.

6. **Example6**. This data set consists of $q = 10$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, \ldots, 1)$ and $(1, 5, 1, 5, 1, 5, 1, 5, 1, 5)$ respectively. For the other cluster the vectors are $(5, 1, \ldots, 1)$ and $(5, 1, 5, 1, 5, 1, 5, 1, 5, 1)$. Table 9.1 shows the results for this problem. We generated 40000 data points, and performed 10-fold cross-validation with 20000 training data and 20000 testing data.

7. **Example7**. This data set consists of $q = 30$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, \ldots, 1)$ and $(10, 5, 10, 5, \ldots, 10, 5)$, respectively. For the other cluster the vectors are $(2, 1, \ldots, 1)$ and $(5, 10, 5, 10, \ldots, 5, 10)$. Table 9.1 shows the results for this problem. We generated 10000 data points, and performed 10-fold cross-validation with 5000 training and 5000 testing data.

8. **Example8**. This data set consists of $q = 30$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, \ldots, 1)$ and $(20, 10, 20, 10, \ldots, 20, 10)$, respectively. For the other cluster the vectors are $(2, 1, \ldots, 1)$ and $(10, 20, 10, 20, \ldots, 10, 20)$. Table 9.1 shows the results for this problem. We generated 10000 data points, and performed 10-fold cross-validation with 5000 training data and 5000 testing data.

9. **Example9**. This data set consists of $q = 50$ input features and $J = 2$ clusters. Both clusters are distributed according to multivariate gaussians. Mean vector and standard deviations for one cluster are $(1, \ldots, 1)$ and $(20, 10, 20, 10, \ldots, 20, 10)$, respectively. For the other cluster the vectors are $(2, 1, \ldots, 1)$ and $(10, 20, 10, 20, \ldots, 10, 20)$. Table 9.1 shows the results for this problem. We generated 10000 data points, and performed 10-fold cross-validation with 5000 training data and 5000 testing data.

10. **OQ data**. See Section 7.3.1 for a description. Table 9.1 shows the results for this data set.

Table 9.1: Average error rates.

	GenProClus	EM	K-Means
Ex1	2.7±0.3	2.5±0.3	11.9±3.3
Ex2	0.9±0.2	0.9±0.2	7.2±0.7
Ex3	7.0±0.4	6.6±0.6	19.2±2.9
Ex4	4.8±0.3	4.5±0.4	35.1±9.4
Ex5	11.4±0.3	5.0±0.4	24.2±0.5
Ex6	0.1±0.06	0.1±0.06	42.2±6.6
Ex7	0.5±0.4	0.9±0.4	49.2±1.8
Ex8	0.6±0.4	0.9±0.4	49.3±1.6
Ex9	0.08±0.1	0.9±0.5	53.2±8.5
OQ	46.9	49.9	47.1

Table 9.2: Average number of iterations.

	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6	Ex7	Ex8	Ex9
GenProClus	5.3	3.9	6.1	5.4	7.2	3.1	3.2	3.2	3.0
K-Means	16.1	10.7	28.5	33.8	16.8	32.7	16.1	15.3	19.4

Table 9.3: GenProClus: Confusion matrix for Example1.

	C0 (input)	C1 (input)
C0 (output)	9917	464
C1 (output)	83	9536

Table 9.4: K-means: Confusion matrix for Example1.

	C0 (input)	C1 (input)
C0 (output)	8364	737
C1 (output)	1636	9263

Table 9.5: GenProClus: Weight values for Example1.

Cluster	Std1	Std2	w_1	w_2
C0	1	4	0.999	0.001
C1	4	1	0.045	0.955

Table 9.6: GenProClus: Confusion matrix for Example2.

	C0 (input)	C1 (input)
C0 (output)	9895	72
C1 (output)	105	9928

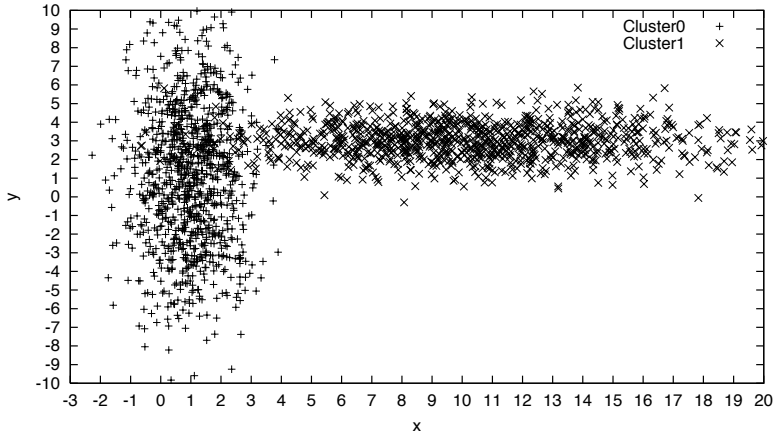

Figure 9.1: Example1: distributions of clusters.

Table 9.7: K-means: Confusion matrix for Example2.

	C0 (input)	C1 (input)
C0 (output)	8578	25
C1 (output)	1422	9975

9.1.2 Results

The performance results reported in Table 9.1 clearly demonstrate the large gain in performance obtained by the GenProClus algorithm against K-means. In particular, the large error rates of K-means for the 10, 30, and 50 dimensional data sets (Examples 6, 7, 8, and 9) show how ineffective a distance function that equally use all input features can be in moderately high dimensional spaces. The gain in performance achieved by locally weighting features is huge in these cases.

GenProClus and EM show similar performances for data sets up to 10 dimensions. As expected, EM performs well on Gaussian distributed data. In higher dimensions (Examples 7, 8,

Table 9.8: GenProClus: Weight values for Example2.

Cluster	Std1	Std2	Std3	w_1	w_2	w_3
C0	1	4	1	0.40	0.02	0.58
C1	1	1	4	0.33	0.66	0.01

Table 9.9: GenProClus: Confusion matrix for Example3.

	C0 (input)	C1 (input)
C0 (output)	9313	705
C1 (output)	687	9295

Table 9.10: K-means: Confusion matrix for Example3.

	C0 (input)	C1 (input)
C0 (output)	7786	1629
C1 (output)	2214	8371

Table 9.11: GenProClus: Weight values for Example3.

Cluster	Std1	Std2	Std3	w_1	w_2	w_3
C0	1	4	1	0.22	0.01	0.77
C1	1	1	4	0.21	0.78	0.01

Table 9.12: GenProClus: Confusion matrix for Example4.

	C0 (input)	C1 (input)
C0 (output)	9688	654
C1 (output)	312	9346

Table 9.13: K-means: Confusion matrix for Example4.

	C0 (input)	C1 (input)
C0 (output)	7249	4265
C1 (output)	2751	5735

Table 9.14: GenProClus: Weight values for Example4.

Cluster	Std1	Std2	Std3	Std4	Std5	w_1	w_2	w_3	w_4	w_5
C0	1	4	1	4	1	0.49	0.02	0.05	0.02	0.42
C1	4	1	1	1	4	0.04	0.45	0.05	0.45	0.01

Table 9.15: GenProClus: Confusion matrix for Example5.

	C0 (input)	C1 (input)	C2 (input)
C0 (output)	8315	0	15
C1 (output)	1676	10000	1712
C2 (output)	9	0	8273

Table 9.16: K-means: Confusion matrix for Example5.

	C0 (input)	C1 (input)	C2 (input)
C0 (output)	9440	4686	400
C1 (output)	411	3953	266
C2 (output)	149	1361	9334

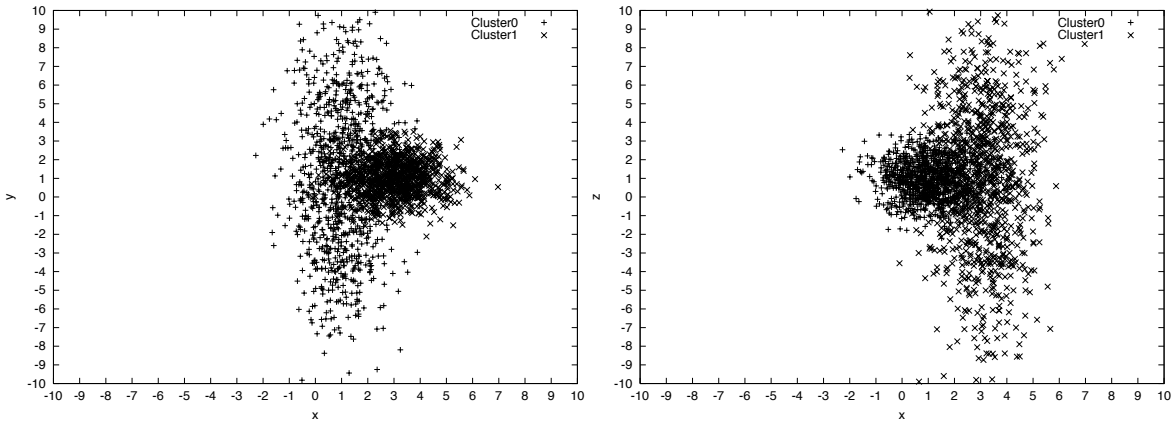

Figure 9.2: Example3: (Left) Distributions of clusters in x-y space. (Right) Distributions of clusters in x-z space.

Table 9.17: GenProClus: Weight values for Example5.

Cluster	Std1	Std2	w_1	w_2
C0	4	1	0.46	0.54
C1	1	4	0.99	0.01
C2	4	1	0.45	0.55

and 9) GenProClus performs consistently better than EM. In particular, in 50 dimensions (Example 9), the error rate of GenProClus is one order of magnitude smaller. In high dimensional spaces, accurate local estimations of full covariance matrices cannot be achieved due to lack of data.

We have also tested the algorithms on a real data set, the OQ data. All three algorithms have large error rates. This shows the high intrinsic difficulty of solving clustering problems. EM has the worst performance in this case. The similar performances of GenProClus and K-means show that we don't gain much by local weightings of features for this specific problem. Large gains are expected in general when clusters are to be discovered in subspaces spanned by different

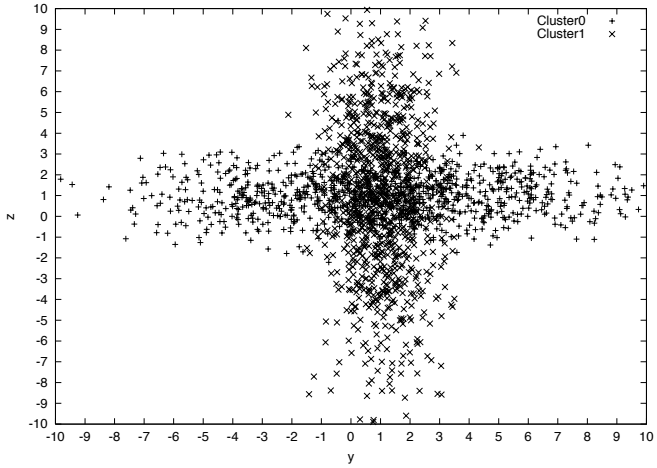

Figure 9.3: Example3: Distributions of clusters in y-z space.

combinations of dimensions.

Table 9.2 shows the average number of iterations performed by GenProClus and K-means to achieve convergence. For each problem, the rate of convergence of GenProClus is at least three times faster (except for Example 5 where is 2.3 times faster); for Example 6 is 10 times faster.

To further test the accuracy of the algorithms, for each problem we have computed the *confusion matrices*. The entry (i, j) in each confusion matrix is equal to the number of points assigned to output cluster i, that were generated as part of input cluster j. We also report the average weight values per cluster obtained over the 10 runs conducted in our experiments. Results are reported in Tables 9.3-9.20.

Tables 9.5, 9.8, 9.11, 9.14, 9.17 and 9.20 show that there is a perfect correspondence between the weight values of each cluster and the correlation patterns of data within the same cluster. This is of great importance for applications that require not only a good partitioning of data, but also information to what features are relevant for each partition. Figures 9.1.1 and 9.3 show the

data distributions of the two clusters of Example 3 projected in the $x-y$, $x-z$, and $y-z$ planes, respectively. We observe that data of cluster 0 are closely correlated in the subspace $x-z$, whereas data of cluster 1 are closely correlated in the subspace $x-y$. Table 9.11 shows that the resulting weight values reflect such local correlations, i.e., larger weights w_1 and w_3 are credited to cluster 0, and larger weights w_1 and w_2 are credited to cluster 1.

As expected, the resulting weight values for one cluster depends on the configurations of other clusters as well. If clusters have the same standard deviation along one dimension i, they receive almost identical weights for measuring distances along that feature. This is informative of the fact that feature i is equally relevant for both partitions. On the other hand, weight values are largely differentiated when two clusters have different standard deviation values along the same dimension i, implying different degree of relevance of feature i for the two partitions (see for example Tables 9.11, 9.14, and 9.17).

9.2 Summary

We have formalized the problem of finding different clusters in different subspaces. Our algorithm discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. This approach avoids the risk of loss of information encountered in global dimensionality reduction techniques.

The output of our algorithm is twofold. It provides a partition of the data, so that the points in each set of the partition constitute a cluster. In addition, each set is associated with a weight vector, whose values give information of the degree of relevance of features for each partition. Our experiments show that there is a perfect correspondence between the weight values of each cluster

and local correlations of data.

We have formally proved that our algorithm converges to a local minimum of the associated error function, and experimentally demonstrated the gain in perfomance we achieve with our method in high dimensional spaces with clusters folded in subspaces spanned by different combinations of features.

Table 9.18: GenProClus: Confusion matrix for Example6.

	C0 (input)	C1 (input)
C0 (output)	9992	11
C1 (output)	8	9989

Table 9.19: K-means: Confusion matrix for Example6.

	C0 (input)	C1 (input)
C0 (output)	5933	4372
C1 (output)	4067	5628

Table 9.20: GenProClus: Weight values for Example6.

Cluster	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
C0	0.21	0.005	0.19	0.005	0.19	0.005	0.19	0.005	0.19	0.01
C1	0.01	0.19	0.005	0.19	0.01	0.20	0.01	0.19	0.005	0.19

Table 9.21: GenProClus: Confusion matrix for Example7.

	C0 (input)	C1 (input)
C0 (output)	2486	13
C1 (output)	14	2487

Table 9.22: K-means: Confusion matrix for Example7.

	C0 (input)	C1 (input)
C0 (output)	1274	1236
C1 (output)	1226	1264

Table 9.23: GenProClus: Confusion matrix for Example8.

	C0 (input)	C1 (input)
C0 (output)	2485	13
C1 (output)	15	2487

Table 9.24: K-means: Confusion matrix for Example8.

	C0 (input)	C1 (input)
C0 (output)	1270	1233
C1 (output)	1230	1267

Table 9.25: GenProClus: Confusion matrix for Example9.

	C0 (input)	C1 (input)
C0 (output)	2497	1
C1 (output)	3	2499

Table 9.26: K-means: Confusion matrix for Example9.

	C0 (input)	C1 (input)
C0 (output)	1227	1245
C1 (output)	1273	1255

Chapter 10

Locally Adaptive Bandwidths for Kernel Density Estimation

In this chapter, we propose a locally adaptive technique to address the problem of setting the bandwidth parameters for kernel density estimation. Our technique is efficient and can be performed in only two data passes. We also show how to apply our technique to efficiently solve range query approximation, classification and clustering problems for large data sets [DG01c].

10.1 Introduction

The application of nearest neighbor techniques on large data sets can be time and resource consuming [BFR98]. In this chapter we introduce an efficient approximation technique based on the observation that the density of the data set contains useful information for both the classification and clustering tasks. For classification, the main point is that, given a query, the values of the

class density functions over the space around it quantify the contribution of the correspondent class within a neighborhood of the query point. The larger the contribution of a given class is, the larger is the likelihood for the query to belong to that class. As for clustering, local maxima of the density function of the data set could represent cluster centers. A hill-climbing procedure applied at a given point, would identify its density attractor.

The same approach can be applied to estimate multidimensional range queries: the value of the integral of the estimated density, computed over the volume defined by the range query, will give an approximation of its selectivity. Answering range queries, is one of the simpler data exploration tasks. However, when the number of dimensions increases, recent results [WSB98] show that the query time is linear to the size of the data set. Thus, the problem of efficiently approximating the selectivity of range queries arises naturally.

For a compact representation of the density of a data set, we use kernel density estimation methods. Kernel methods pose the problem of setting the bandwidth parameters. Current work on this problem in statistics has addressed only the one dimensional case satisfactorily [Sco92]. Approximately optimal bandwidth parameters in the multi-dimensional case have been obtained only for the special case in which the following conditions are all true: (i) the attributes are independent, (ii) the distribution along each dimension is Gaussian and (iii) all bandwidths, for all kernels and dimensions, are to be set to the same value [Sco92]. For example, one solution to the problem of computing the bandwidths is given by Scott's rule [Sco92], which estimates one bandwidth parameter per attribute, by setting its value to a quantity proportional to the standard deviation of the sample on that attribute. The rule assumes attribute independence.

To achieve more accurate results we propose an adaptive bandwidth estimation technique that adapts the bandwidth of the kernels using local information, and does not assume independence between the attributes. The setting of the bandwidth for each kernel is based on the extension of the points in the neighborhood, and each kernel uses the same bandwidth for all dimensions. Our technique is efficient and can be performed in only two data passes.

Using the range query approximation problem as a benchmark, we show the performance improvement we achieve with our method over Scott's rule by using a variety of both synthetic and real data sets. We then show how to apply our technique to efficiently solve classification and clustering problems. We focus on classification and show the results we obtained on simulated data sets.

10.2 Related Work

Locally adaptive density estimators have been introduced in multivariate statistics. The *balloon* estimator [TS92, Sai99] varies the bandwidth at each estimation point. Given a point $\mathbf{x}$ at which the density is to be estimated, the bandwidth value is set to the distance $h_k(\mathbf{x})$ of $\mathbf{x}$ from the Kth nearest data point. Then, kernels of the same size $h_k(\mathbf{x})$ are centered at each data point, and the density estimate is computed by taking the average of the heights of the kernels at the estimation point.

The balloon estimator requires that all data points are kept in memory, since the bandwidth value depends on the estimation point $\mathbf{x}$, and on its distance from the Kth nearest data point (unless an approximation scheme is used). Furthermore, K acts as a smoothing parameter, and its setting is critical. The computation of a proper value for K is an open problem, and expensive least-squares

cross-validation techniques are used to determine its value.

These limitations, along with the fact that a different density approximation function is computed for each estimation point, make this approach not suited for efficient solutions of data exploration tasks considered here.

Another locally adaptive technique is the *sample-point* estimator [BMP77, Sai99]. It places a kernel at each data point $\mathbf{x}$. Each kernel has its own bandwidth, set to the distance of the center $\mathbf{x}$ from the Kth nearest point. Again, as for the balloon estimator, the choice of K is critical, and the problem of setting its value is open. We also observe that both the balloon and sample-point estimators fit a kernel at each data point. It is not clear how to couple these techniques with sampling, since the bandwidth values won't properly adjust to the sample size. In contrast, in our algorithm bandwidth values are function of both sample and original data set sizes.

10.3 The Range Query Approximation Problem

Let R be a data set of N points, each with q real attributes. The domain of each attribute is scaled to the real interval $[0, 1]$. We consider range queries of the form $(a_1 \leq R.A_1 \leq b_1) \wedge \ldots \wedge (a_q \leq R.A_q \leq b_q)$. The *selectivity* of a range query Q, $sel(R, Q)$, is the number of points in the interior of the hyper-rectangle it represents. Since N can be very large, the problem of approximating the selectivity of a given range query Q arises naturally. Approaches proposed to address this problem include multidimensional histograms [IP99, GKT+00], kernels [SFB99, GKT+00], and wavelets [VWI98, CGR+00].

To formalize the notion of approximating the selectivity of range queries, let $f(x_1, \ldots x_q)$

be a q-dimensional, non-negative function, defined in $[0,1]^q$ and with the property $\int_{[0,1]^q} f(x_1, \ldots x_q)$ $dx_1 \ldots dx_q = 1$. f is a *probability density function*. The value of f at a specific point $\mathbf{x} = (x_1, \ldots x_q)$ is the limit of the probability that a tuple exists in area U around $\mathbf{x}$ over the volume of U, when U shrinks to $\mathbf{x}$. Then, for a given such f, to find the selectivity of a query, we compute the integral of f in the interior of the given query Q: $sel(f, Q) = \int_{[a_1,b_1]\times\ldots\times[a_q,b_q]} f(x_1, \ldots x_q) dx_1 \ldots dx_q$.

For a given R and f, f is a good *estimator* of R with respect to range queries if for any range query Q, the selectivity of Q on R and the selectivity of Q on f multiplied by N are similar. To formalize this notion, we define the following error metrics (also used in [VWI98]).

Following [VWI98], we define the *absolute error* of a given query Q to be simply the difference between the real value and the estimated value: $\epsilon_{abs}(Q, R, f) = |sel(R, Q) - N\ sel(f, Q)|$. The *relative error* of a query Q is generally defined as the ratio of the absolute error over the selectivity of the query. Since in our case a query can be empty, we follow [VWI98] in defining the relative error as the ratio of the absolute error over the maximum of the selectivity of Q and 1:

$$\epsilon_{rel}(Q, R, f) = \frac{|sel(R, Q) - N\ sel(f, Q)|}{\max(1, sel(R, Q))}.$$

10.4 Multi-Dimensional Kernel Density Estimators

All the proposed techniques for approximating the query selectivity compute a density estimation function. Such function can be thought as an approximation of the probability distribution function, of which the data set at hand is an instance. It follows that statistical techniques which approximate a probability distribution [Sco92, WJ95], such as kernel estimators, are applicable to address the query estimation problem.

For a data set R, let S be a set of tuples drawn from R at random. Assume there exists a q dimensional function $k(x_1, \ldots, x_q)$, the *kernel function* [1], with the property $\int_{[0,1]^q} k(x_1, \ldots, x_q)$ $dx_1 \ldots dx_q = 1$. The approximation of the underlying probability distribution of R is $f(x_1, \ldots, x_q) = \frac{1}{N} \sum_{t_i \in S} k(x_1 - t_{i_1}, \ldots, x_q - t_{i_q})$, and the estimation of the selectivity of a q-dimensional range query Q is $sel(f, Q) = \int_{[0,1]^q \cap Q} f(x_1, \ldots, x_q) dx_1 \ldots dx_q = \frac{1}{N} \sum_{t_i \in S} \int_{[0,1]^q \cap Q} k(x_1 - t_{i_1}, \ldots, x_q - t_{i_q}) dx_1 \ldots dx_q$. It has been shown that the shape of the kernel function does not affect the approximation substantially [Cre93]. The key feature is the standard deviation of the function, or, its bandwidth. Therefore, we choose a kernel function that is easy to integrate, i.e. the q-dimensional Epanechnikov kernel function [Cre93], whose equation centered at 0 is:

$$k(x_1, \ldots, x_q) = (\frac{3}{4})^q \frac{1}{B_1 B_2 \ldots B_q} \prod_{1 \leq i \leq q} (1 - (\frac{x_i}{B_i})^2)$$

if $|\frac{x_i}{B_i}| < 1$ for all i, and 0 otherwise (Figure 10.4). The q parameters $B_1, \ldots, B_q$ are the bandwidth

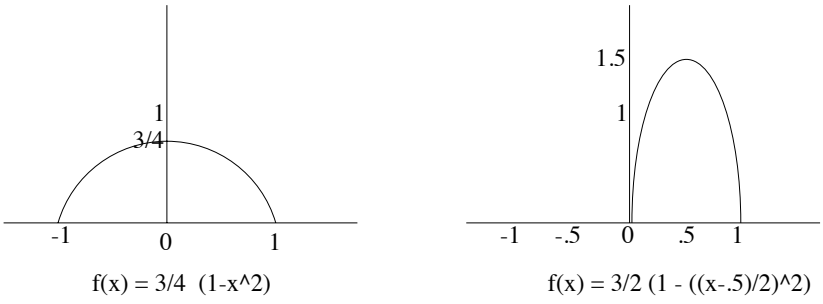

Figure 10.1: The one-dimensional Epanechnikov kernel, with $B = 1$, centered around the origin, and with $B = 1/2$, centered at 0.5.

of the kernel function along each of the q dimensions. The magnitude of the bandwidth controls how far from the sample point we distribute the weight of the point. As the bandwidth becomes

[1] This definition of *kernel function* is different from the one given in the context of SVMs, although both functions serve as a measure of similarity between points.

smaller, also the non-zero diameter of the kernel becomes smaller.

To estimate the bandwidths, typically Scott's rule [Sco92] is used:

$$B_i = \sqrt{5}\, s_i\, |S|^{-\frac{1}{q+4}},$$

where s_i is the standard deviation of the sample on the i-th attribute. This rule is derived using the assumption of Gaussian data distribution, therefore in general it oversmoothes the actual underlying function. The rule assumes attribute independence. Other approaches for setting the bandwidths, such as one-dimensional least squares cross-validation, also assume attribute independence [PT92].

10.4.1 Computing the Selectivity

In [GKT+00] we have shown how to use multi-dimensional kernel density estimators to efficiently address the multi-dimensional range query selectivity problem. We used Scott's rule for setting the bandwidths. We presented an experimental study that shows performance improvements over traditional techniques for density estimation, including sampling [HS92], multi-dimensional histograms [PI97], and wavelets [VWI98]. The main advantage of kernel density estimators is that the estimator can be computed very efficiently in one data pass, during which we both sample the data set and approximate the standard deviation along each attribute.

Since the q-dimensional Epanechnikov kernel function is the product of q one-dimensional degree-2 polynomials, its integral within a rectangular region can be computed in $O(q)$ time:

$sel(f, [a_1, b_1] \times \ldots \times [a_q, b_q]) =$

$\frac{1}{N} \int_{[a_1,b_1]\times\ldots\times[a_q,b_q]} (\sum_{1\le i\le |S|} k_i(x_1, \ldots, x_q) dx_1 \ldots dx_q) =$

$\frac{1}{N} \int_{[a_1,b_1]\times\ldots\times[a_q,b_q]} \sum_{1\le i\le |S|} (\frac{3}{4})^q \frac{1}{B_1 B_2 \ldots B_q} \prod_{1\le j\le q} (1 - (\frac{x_j - X_{ij}}{B_j})^2)\, dx_1 \ldots dx_q =$

$\frac{1}{N}(\frac{3}{4})^q \frac{1}{B_1 B_2 \ldots B_q} \sum_{1 \leq i \leq |S|} \int_{[a_1,b_1]}(1 - (\frac{x_1 - X_{i1}}{B_1})^2)\, dx_1 \ldots \int_{[a_q,b_q]}(1 - (\frac{x_q - X_{iq}}{B_q})^2)\, dx_q$.

It follows that, for a sample of $|S|$ tuples, $sel(f, Q)$ can be computed in $O(q|S|)$ time.

10.5 Locally Adaptive Bandwidths

Kernel-based methods are nearest-neighbor-type algorithms: to obtain the density estimate at a given point, assuming far-off points have negligible contribution to the sum, one has to consider only the kernel contributions of the nearest neighbors. It is therefore reasonable to adapt the bandwidths of a kernel centered at a specific point according to the extension of the neighborhood of its center, so that the kernel will mainly contribute to the density estimation of points within that same local neighborhood. This allows us to take into account local attribute correlations: kernels with more points close to them (according to the L_2 distance metric) will have smaller bandwidths than those with fewer points close to them. Real life data often present correlations among attributes, and therefore performance benefits from this approach [Sco92].

As a consequence, we develop a heuristic (ADAptive BANDwidth) that locally adapts the bandwidths of kernels, according to the extension of points within the kernel neighborhood. AdaBand uses the same bandwidth for all the dimensions of a given kernel, but changes the bandwidth from kernel to kernel. The heuristic works as follows.

A uniform random sample S of a given size, say $|S|$, is first produced. Let R be the original data set, and $|R|$ its size. Each point in S distributes its weight over the space around it. We want each kernel to distribute its weight over an equal number of points in the data set, i.e. as many points as $\frac{|R|}{|S|}$. For each point $\mathbf{s} \in S$, we compute its distance from the data points in R. Among

these $|R|$ distances, we identify the one that corresponds to the $\frac{1}{|S|}$-quantile, i.e. the distance at position $\lceil \frac{|R|}{|S|} \rceil$ in the sorted sequence of distances. Let D be such quantile. D can be seen as the distance of $\mathbf{s}$ from the vertex of the hypercube centered at $\mathbf{s}$ that includes a neighborhood of $\lceil \frac{|R|}{|S|} \rceil$ points. To set the bandwidth B of a kernel centered at $\mathbf{s}$, we compute the projection of D along each dimension (and double it to avoid possible uncovered areas that may contain a fraction of the $\lceil \frac{|R|}{|S|} \rceil$ points), resulting in $B = \frac{2 \times D}{\sqrt{q}}$. Each kernel has one bandwidth value B associated with it, valid for all the q dimensions. The algorithm, therefore, stores $(q + 1)$ numbers per kernel: q values for the coordinates of the center, and one value for the bandwidth. Figure 10.2 gives the outline of the algorithm.

For comparison purposes, we have also performed experiments in which we estimate a bandwidth value for each dimension and each kernel by using a localized standard deviation at the kernel's center along each dimension. In this case we store $2q$ numbers per kernel, and therefore the sample size is reduced to $\frac{EstSize}{2q}$, where EstSize is the size of the estimator. We have observed that the loss due to the reduced sample size overcomes the gain achieved by storing a distinct bandwidth value for each dimension. AdaBand, instead, seems to capture sufficient local information by storing one bandwidth per kernel, without over penalizing the sample size.

10.5.1 Running Time

Computing a kernel density estimator with $|S|$ kernels, as described above, can be done in two data passes. During the first pass, a random sample of size $|S|$ is taken. During the second pass, an approximation of the $\frac{1}{|S|}$-quantiles for the points in S is computed.

In the implementation of AdaBand, to efficiently estimate the quantiles we use the tech-

Input: A q-dimensional data set R with N points, and parameter $EstSize$ (Estimator Size):

1. Set $KCount = \frac{EstSize}{q+1}$;
2. Take a random sample S of size $KCount$;
3. For each point $\mathbf{s} \in S$:
   (a) Compute the N distances $d_1, \ldots, d_N$ of $\mathbf{s}$ from the points in R;
   (b) Compute the $\frac{1}{KCount}$-quantile D of the distances $d_1, \ldots, d_N$;
   (c) Set $B_i = \frac{2 \times D}{\sqrt{q}}$, for $i = 1, \ldots, q$, where B_i is the bandwidth along dimension i of the kernel centered at $\mathbf{s}$.

Figure 10.2: The AdaBand algorithm

nique described in [MRL98], which guarantees arbitrarily tight error bounds and, for a given desirable accuracy, allows the estimation of the optimal space complexity.

10.6 Classification

Here we show how we can apply the AdaBand algorithm to address classification problems. In a classification problem we are given J classes and N training observations. The training observations consist of q attribute measurements $\mathbf{x} = (x_1, \ldots, x_q) \in \Re^q$ and the known class labels: $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $y_i \in \{1, \ldots, J\}$. The objective is to predict the class label of a given query $\mathbf{x}_0$. The given training data are used to obtain the estimates.

We assume, again, that the given data set is large, and we want to be able to perform our prediction efficiently. The AdaBand algorithm, applied within the data of each class, allows to efficiently compute an estimation of the class density distributions of the given data set. Formally, by denoting with S_j the sample extracted from the N_j training data of class j, the within class density function estimate at a given point $\mathbf{x}$ is

$$\hat{f}_j(\mathbf{x}) = \frac{1}{N_j} \sum_{i \in S_j} (\frac{3}{4})^q \frac{1}{{B_i}^q} \prod_{1 \leq j \leq q} (1 - (\frac{x_j - s_{ij}}{B_i})^2).$$

For a given query point $\mathbf{x}_0$, we have then J within class density function estimates: $\hat{f}_1(\mathbf{x}_0), \ldots,$ $\hat{f}_J(\mathbf{x}_0)$. The class j^* that gives the largest value $\int_I \hat{f}_j(\mathbf{x}_0) d\mathbf{x}_0$, computed over an interval I centered at $\mathbf{x}_0$, is our prediction for the class of $\mathbf{x}_0$, i.e.,

$$j^* = \arg \max_{1 \leq j \leq J} \int_I \hat{f}_j(\mathbf{x}_0) d\mathbf{x}_0.$$

We observe that the integrals can be computed efficiently in $O(q|S_j|)$ time, as described in section 10.4.1. The integral operation allows to smooth away the estimated density functions, thereby achieving more accurate results than with a pointwise estimation. This method, which we call DenClass, can be seen as an attempt to approximate the optimal Bayesian classification error rate, where $\hat{P}(j|\mathbf{x}_0) = \int_I \hat{f}_j(\mathbf{x}_0) d\mathbf{x}_0$ is our density based approximation of class posterior probabilities at query points. We note that, for a correct class assignment, the classifier$\hat{f}_j(\mathbf{x})$ needs only to preserve the order relation among the estimated quantities. This means that we can afford biased estimates, as long as all are affected roughly in the same proportion. The experimental results indeed suggest that the integration operation conveys robustness to our method. The length of the interval I is an input parameter of the DenClass algorithm, which we optimize by cross-validation in our experiments.

The DenClass algorithm is also related to the procedure introduced in [FF99]. The authors in [FF99] discuss a *bump hunting* method that seeks sub-regions of the space of input values within which the average value of the target is much larger than its average over the entire input space. This approach can be used for function approximation, classification, and clustering. For classification, the goal is to identify those regions within which an observation is most likely to be from one specific class j. These are the regions where $P(j|\mathbf{x})$ is larger than that of any other class. Similarly, our method classifies the query point $\mathbf{x}_0$ with the label of the class whose density function gives the largest "bump" contribution within a region centered at $\mathbf{x}_0$. Figure 10.3 gives the outline of the DenClass algorithm.

Classifier construction: Given a q-dimensional data set R with N points and J classes, and the input parameter $|S_j|$ for each class $j \in J$:

1. Run AdaBand on R_j and $EstSize = |S_j|(q+1)$ (R_j is the set of points in R with label j).

Output: $\hat{f}_j(\mathbf{x})$, for $j = 1, \ldots, J$.

Testing phase: Given a query point $\mathbf{x}_0$:

1. Classify $\mathbf{x}_0$ to class j^* s.t.

   $j^* = \arg\max_{1 \le j \le J} \int_I \hat{f}_j(\mathbf{x}_0) d\mathbf{x}_0$.

Figure 10.3: The DenClass algorithm

Density Estimation: Given a q-dimensional data set R with N points, and the input parameter $|S|$:

1. Run AdaBand on R and $EstSize = |S|(q+1)$.

Output: $\hat{f}(\mathbf{x})$.

Clustering phase: Given a query point $\mathbf{x}_0$, and the input parameter t:

1. Compute the gradient $\nabla\hat{f}(\mathbf{x}_0)$;
2. Perform hill-climbing. Let $\mathbf{x}^*$ be the resulting density attractor;
3. If $\hat{f}(\mathbf{x}^*) > t$, assign $\mathbf{x}_0$ to the cluster identified by $\mathbf{x}^*$;
4. If $\hat{f}(\mathbf{x}^*) \leq t$,
   (a) Use a grid to merge $\mathbf{x}_0$ with a connected cluster center $\mathbf{x}^{*'}$;
   (b) Assign $\mathbf{x}_0$ to the cluster identified by $\mathbf{x}^{*'}$.

Figure 10.4: The DenClust algorithm

10.7 Clustering

The method we present here is an extension of [HK98]. The technique discussed in [HK98] employs all data points; a grid approximation is proposed to cope with large data sets, and the resulting complexity depends on the size of the grid.

The DenClass algorithm can be extended to also address clustering problems. In clustering, data are unlabelled, and the density estimation is conducted for the whole data set. Local

maxima of $\hat{f}(\mathbf{x})$, that are above a certain threshold t, can be considered cluster centers. A hill-climbing procedure applied at a given point identifies its density attractor. Points that converge to the same attractor belong to the same cluster. For density attractors below t, we can use connectivity information (using a grid in input space, for example) to merge them with connected cluster centers.

What is a good choice for t? If we assume that the data set R is noise-free, all density attractors $\mathbf{x}^*$ for S are significant and t should be chosen in $0 \leq t \leq \min_{\mathbf{x}^*}\{\hat{f}(\mathbf{x}^*)\}$. In most cases the data set will contain noise. If the noise level can be modeled by taking into account knowledge specific to the problem at hand, then t should be chosen above such level. As an alternative, the value of t could be set above the average value of the density function evaluated at the attractors: $\frac{1}{|\mathbf{x}^*|}\sum_{\mathbf{x}^*}\hat{f}(\mathbf{x}^*)$. In general, the smaller the value of t is, the more sensitive the clustering algorithm will be to outliers; the larger t is, the less details will be captured by the algorithm. Figure 10.4 gives the outline of the method, which we call DenClust.

Chapter 11

Locally Adaptive Bandwidths: Experimental Evaluation

This chapter validates the efficiency and accuracy of locally adapting the bandwidth parameters for kernel density estimation, for both range query approximation and classification problems.

11.1 Experimental Settings

In our experiments we compare the performance of AdaBand, Scott's rule and Random Sampling on synthetic and real-life data with real valued attributes. For both AdaBand and Scott's rule we use the formulas described in section 10.4.1 to compute the selectivity of range queries. We examine the behavior of the methods as additional space for storing the estimator becomes available. We also evaluate the accuracy of the methods as the dimensionality of data increases.

To test the accuracy of the DenClass algorithm we use simulated data sets and compare its performance with well known methods in the literature: K-NN, C4.5 decision tree [Qui93], and K-means. We include K-means since it allows a compact representation of the data set, specifically the within class mean vectors. Note that since we are applying K-means to classification problems the value of K is equal to the number of classes. We also compare the performance of DenClass with an algorithm (that we call DenScott) that proceeds as DenClass, but sets the bandwidth values according to Scott's rule (one bandwidth value for each dimension and for each class). Procedural parameters ($|I|$ for DenClass and DenScott, and K for K-NN) are determined empirically through cross-validation.

11.2 Simulated Data

We have designed the following synthetic data sets for the range query selectivity problem. In Figures 11.1- 11.3 the 1-norm average relative errors computed over five runs are reported.

OneGaussian: This data set contains 10^6 5-dimensional points drawn according to a Gaussian distribution with standard deviation set to 5 along each dimension. In this situation Scott's rule finds the optimal bandwidth values.

MultiGaussian. This data set contains 10^6 10-dimensional points drawn according to 25 Gaussian distributions with mean values randomly chosen within the range $[25, 74]$, and standard deviation values for all dimensions set to 0.25. Each Gaussian generates the same number of data points.

DiffGaussian: This data set contains 10^6 10-dimensional points equally drawn according to 25 Gaussian distributions with mean values randomly chosen within the range $[25, 74]$, and standard deviation values randomly chosen within the set $\{0.1, 0.2, 0.3, \ldots, 1.0\}$.

NoisyGaussian: This data set contains 10^6 10-dimensional points. 25% of the data ($250,000$ points) is uniformly distributed random noise. The remaining $750,000$ points are equally generated according to 25 Gaussian distributions with mean values randomly chosen again within the range $[25, 74]$, and standard deviation values for all dimensions set to 0.25.

The following data sets are used for the classification problem. For each of them, five independent training data were generated. For each of these, an additional test set (of size 2,000 for Ex1, 1,000 for Ex2, 6,000 for Ex3, 2,000 for Ex4, and 3,000 for Ex5) was generated. Error rates and standard deviation values are computed over all such classifications and reported in Table 11.1.

Example 1: This data set has $q = 5$ attributes, $N = 500,000$ data points, and $J = 2$ classes (250,000 points per class). The data for both classes are generated from a multivariate normal distribution with standard deviation 8 along each dimension, and mean vector $(40, \ldots, 40)$ in one case, and $(60, \ldots, 60)$ in the other. The sample size used for the DenClass algorithm is $|S_j| = 500$ for both classes. By taking into account both the sample points and the bandwidth values, the resulting classifier $\hat{f}_j(\mathbf{x})$, $j = 1, 2$, requires the storage of 6,000 numbers. We therefore allow a sample of 600 points for both classes for the other four methods.

Example 2: This data set has $q = 2$ attributes, $N = 500,000$ data points, and $J = 2$ classes (250,000 points per class). The data for this problem are generated as in the previous example, with the addition of 25% uniformly distributed random noise. We use $|S_j| = 500$ for DenClass, and accordingly a sample size of 750 points for the other methods.

Example 3: This data set has $q = 2$, $N = 240,000$, and $J = 2$ classes. Each class contains six spherical bivariate normal subclasses, having standard deviation one. The means of the 12 subclasses are chosen at random without replacement from the integers $[25+2k]_{k=0}^{24} \times [25+2k]_{k=0}^{24}$.

For each class, data are evenly drawn from each of the six normal subclasses. We use $|S| = 1,000$ for DenClass for this example, and accordingly a sample size of 1,500 data points per class for the other methods.

Example 4: This data set has $q = 10$, $N = 400,000$, and $J = 2$ classes. 20% of the N points is uniformly distributed random noise. The data for both classes are generated from a multivariate normal distribution with standard deviation 9 along each dimension, and mean vector $(40, \ldots, 40)$ in one case, and $(50, \ldots, 50)$ in the other.

Example 5: This data set has $q = 10$, $N = 600,000$, and $J = 3$ classes (200,000 points per class). 20% of the N points is uniformly distributed random noise. The data for all three classes are generated from a multivariate normal distribution with standard deviation 10 along each dimension. The mean vectors are: $(40, \ldots, 40)$, $(50, \ldots, 50)$, and $(60, \ldots, 60)$.

11.3 Real Data

We use three real data sets. The USCities and the NorthEastern data sets contain, respectively, 1,300,000 postal addresses of cities in the US, and 130,000 postal addresses of the North Eastern states. Each point has two attributes. We also use the Forest Cover Data set from the UCI KDD archive. This data set was obtained from the US Forest Service (USFS). It includes $590,000$ points, and each point has 54 attributes, 10 of which are numerical. In our experiments we use the entire set of 10 numerical attributes. In this data set the distribution of the attributes is non-uniform, and there are correlations between pairs of attributes. In Figures 11.3-11.3 the 1-norm average relative errors computed over five runs are reported.

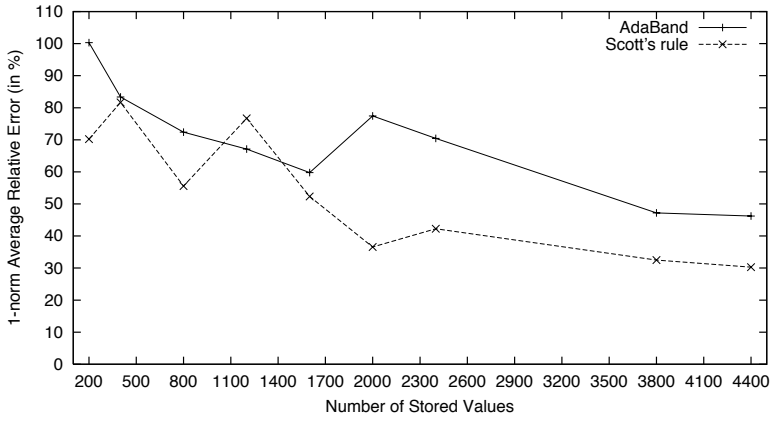

Figure 11.1: OneGaussian data set, Query workload 3, 5-dim.

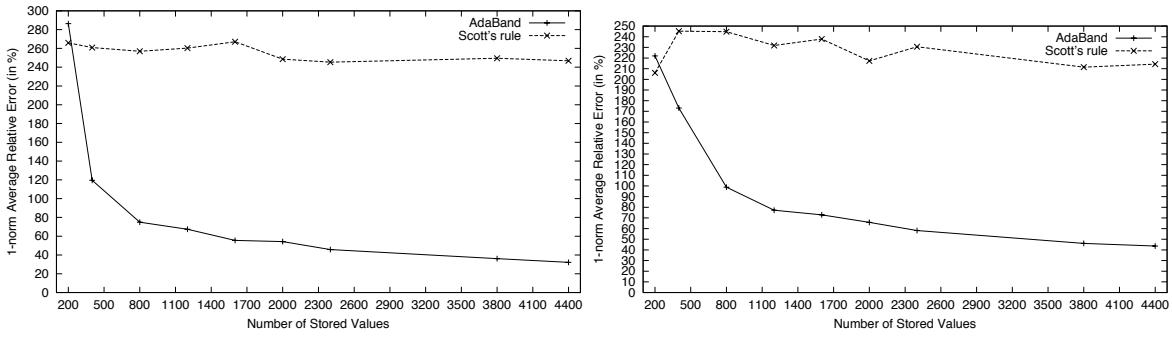

Figure 11.2: (Left) MultiGaussian data set , Query workload 2, 10-dim. (Right) DiffGaussian data set, Query workload 2, 10-dim.

11.4 Query Workloads

To evaluate the techniques on the range query approximation problem we generated workloads of three types of queries.

Workloads 1 and 2 contains 10^4 random queries with selectivity approximately 10% and 1%, respectively. Workload 3 consists of $20,000$ queries of the form $(R.A_1 < a_1) \wedge \ldots \wedge (R.A_q < a_q)$, for a randomly chosen point $(a_1, \ldots, a_q) \in [0,1]^d$.

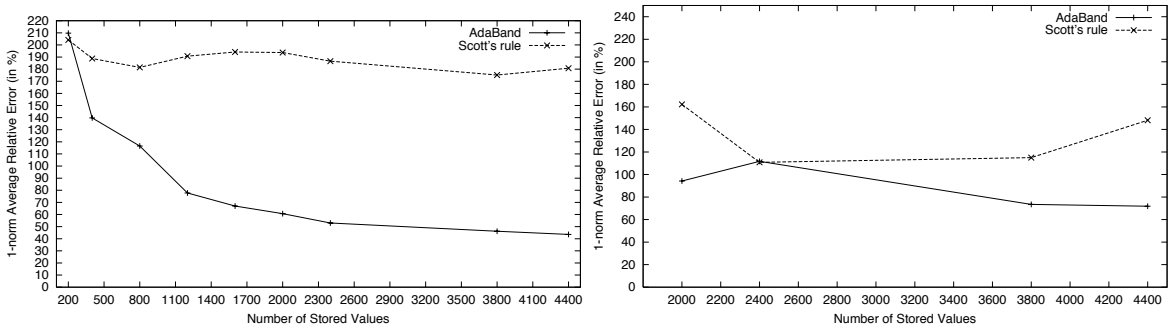

Figure 11.3: NoisyGaussian data set, 10-dim. (Left) Query workload 2. (Right) Query workload 3.

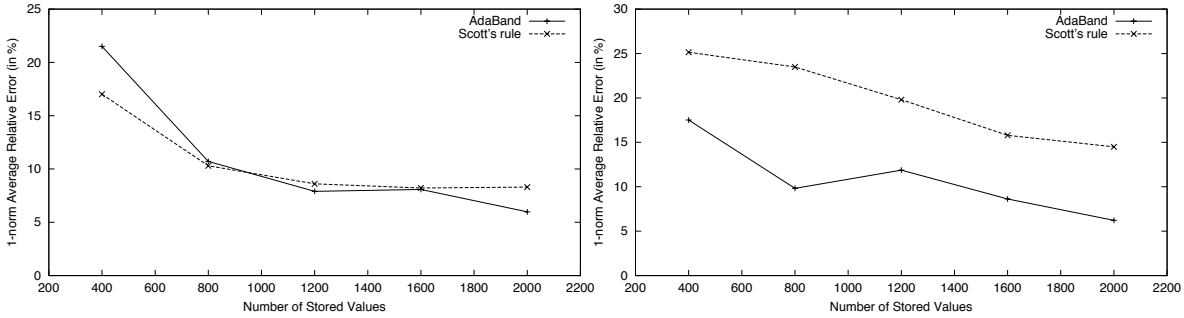

Figure 11.4: (Left) NorthEastern data set, Query workload 1, 2-dim. (Right) NorthEastern data set, Query workload 3, 2-dim.

For each workload we compute the average absolute error $\| e_{abs} \|_1$ and the average relative error $\| e_{mod} \|_1$.

11.5 Experimental Results for Query Approximation

The OneGaussian data set has been designed to test AdaBand performance under optimal conditions for Scott's rule. Scott's rule finds optimal bandwidth values for this data set. Figure 11.1 shows the results for query workload 3. As expected, Scott's rule shows the best performance, but AdaBand is not too far from it. This means that we don't loose too much in performance with our

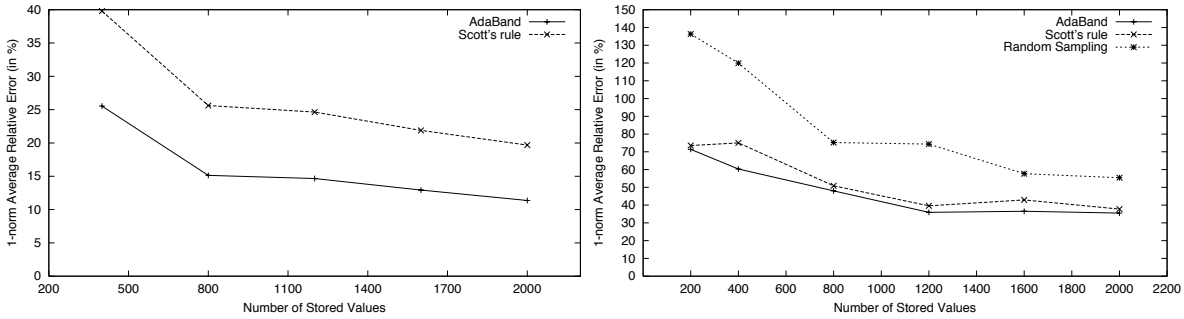

Figure 11.5: (Left) USCities data set, Query workload 3, 2-dim. (Right) Forest Cover data set, Query workload 1, 10-dim.

adaptive technique in the ideal case for Scott's rule.

The variance (spikes) observed in Figure 11.1 for smaller estimator sizes may be due to the fact that the data set is 5-dimensional, and therefore larger sample sizes are required to attain a smoother performance behavior. Furthermore, workload 3 presents a higher degree of difficulty since queries in this case have arbitrary sizes. This characteristic may also have contributed to the variance of the performance.

Figures 11.3-11.3 show the results for the MultiGaussian, DiffGaussian and NoisyGaussian data sets on query workloads 2 and 3. We obtained similar results for the MultiGaussian and DiffGaussian data sets. AdaBand outperforms by far Scott's rule in both cases. Scott's rule is not able to scale its performance as the size of the estimator increases, whereas our technique is capable of adapting the bandwidths according to the number of kernels that become available. A similar behavior is observed for the NoisyGaussian data set on query workload 2 (Figure 11.3, left). On query workload 3, the increase in error of Scott's rule is likely to be due to the fact that the NoisyGaussian data set is 10-dimensional, and workload 3 includes queries of arbitrary sizes. Indeed, errors on

Table 11.1: Average classification error rates and standard deviation values.

Method	Ex1	Ex2	Ex3
DenClass	**0.3**$\pm$ 0.1	15.4 $\pm$ 0.1	**0.3** $\pm$ 0.6
K-NN	0.4 $\pm$ 0.1	**15.3**$\pm$ 0.1	**0.3** $\pm$ 0.6
DenScott	10.4 $\pm$ 12.4	46.4 $\pm$ 1.4	0.6 $\pm$ 0.9
C4.5	2.5 $\pm$ 0.5	17.0 $\pm$ 0.4	0.6 $\pm$ 0.7
K-means	0.4 $\pm$ 0.1	15.6 $\pm$ 0.2	26.7 $\pm$ 8.7

different runs showed a large variance for the tested estimator sizes. The plot for AdaBand is rather flat, but does show improvement for larger estimator sizes.

Figures 11.3-11.3 show the results for the real data sets. AdaBand shows large improvements in performance over Scott's rule with both the NorthEastern and USCities data sets for query workload 3. For query workload 1 AdaBand performs slightly better than Scott's rule on the NorthEastern and Forest Cover data sets. Figure 11.3 also shows, for comparison, the results obtained applying Random Sampling, that gives the worst performance.

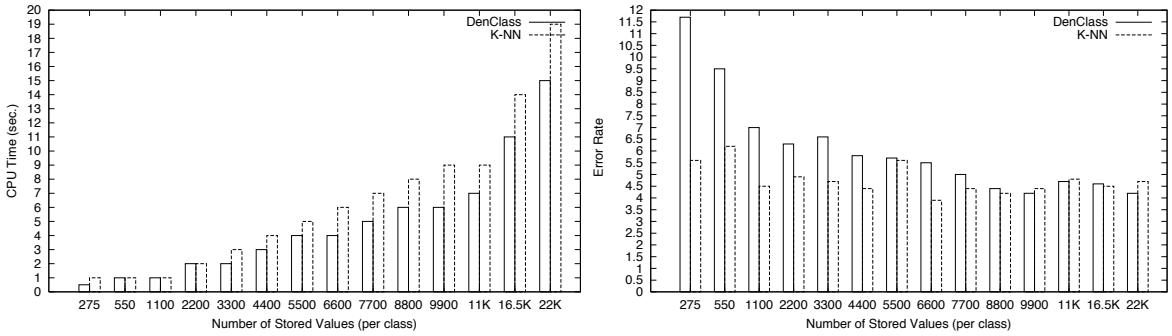

Figure 11.6: Example 4, 10-dim, two classes. (Left) CPU-time versus Number of Stored Values. (Right) Error Rate versus Number of Stored Values.

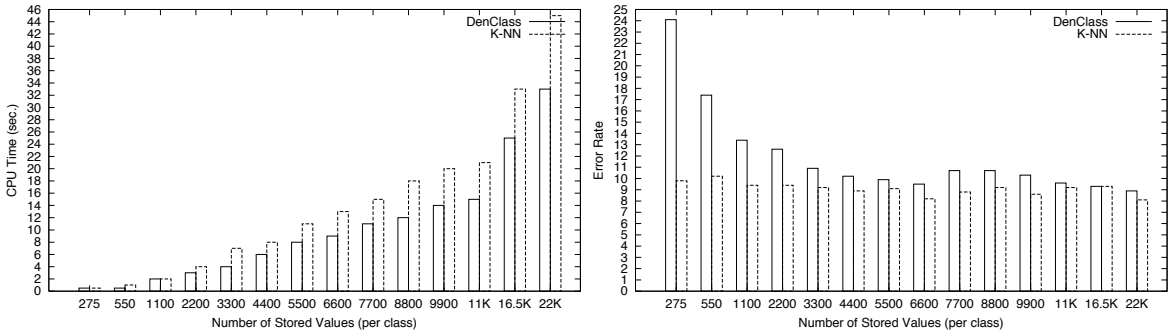

Figure 11.7: Example 5, 10-dim, three classes. (Left) CPU-time versus Number of Stored Values. (Right) Error Rate versus Number of Stored Values.

11.6 Experimental Results for Classification

Table 11.1 shows the error rates obtained for classification. We observe that DenClass outperforms DenScott, C4.5 and K-means in all three cases. DenScott shows a high sensitivity to uniformely distributed noise (Example 2). This is likely due to the global nature of the settings of bandwidth values. In general, the error rates for the DenScott procedure suffer from large variance. This result demonstrates the lack of robustness of techniques based on Scott's rule for classification purposes, and shows the superiority of our local method DenClass.

DenClass and K-NN show similar performances in each problem in Table 11.1. Figures 11.5-11.5 plot CPU-times and Error Rates versus the Number of Stored Values for both DenClass and K-NN, and for Examples 4 and 5 respectively. In both cases we observe that, as the number of Stored Values increases, DenClass is capable of approximating K-NN in accuracy, while significantly improving the execution time. These results provide evidence that we have successfully designed an efficient approximation scheme for nearest neighbor approaches to classification. Such approximation makes K-NN techniques applicable in large data sets. Given that nearest neigh-

bor methods in many benchmark studies turn out to be competitive, and often are among the best performers, an efficient approximation that allows its usage for large data sets is indeed highly desirable.

11.7 Related Work

Multi-dimensional histograms are particularly suited as density estimators when each attribute has a finite discrete domain. Efficient construction of accurate histograms becomes a problem in high dimensional spaces and when the attributes are real valued. In such cases, in fact, histogram constructions become inefficient [GKT+00]. In contrast, our locally adaptive kernel approach allows an efficient estimator construction that requires only two data passes. Efficient query approximation can be performed in time linear to the size of the estimator and to the dimensionality. Furthermore, kernel density estimators have sufficient expressive power, since any distribution can be represented as the sum of a sufficient number of kernel contributions. As a consequence, they are able to provide accurate estimators.

In [BFG99] the density function of the data is estimated in order to build a clustered index for efficient retrieval of approximate nearest neighbor queries. Both our density estimation approach and the clustering process in [BFG99] work on all dimensions simultaneously. The data density modeling is performed in the two cases for different purposes. In [BFG99], the model of the density is used to reorganize the data on the disk, with the objective of minimizing the number of cluster scans at query time. In our case it synthesizes the relevant information about the data to directly address the tasks.

Furthermore, the density estimation process itself is different. In [BFG99], the location

in space for placing the Gaussian kernels is determined by finding clusters in the data. We instead extract a uniform random sample from the data, and center the kernels at the sampled points. As a consequence, in our case the number of kernels used is driven by the estimator size we can afford. In [BFG99], the number of clusters used affects the amount of data to be scanned at query time, and its "optimal" value needs to be estimated.

Our approach is related to the Variable-kernel Similarity Metric (VSM) technique, introduced in [Low95]. Here, the K-nearest neighbor technique is combined with a variable kernel method to address classification problems. The bandwidth of a Gaussian kernel centered at a point $\mathbf{x}$ is set proportionally to the average distance from $\mathbf{x}$ of the k neighbors. The classification problem for a given query point is then solved by taking the weighted average of the known correct outputs of the k nearest neighbors of the query point. The weight values are provided by the kernel, based on the distance of each neighbor from the query point. Distances are also weighted using global weights computed by mean of a cross-validation procedure, and the conjugate gradient optimization method.

The VSM technique requires, for each given query point, the computation of the k-nearest neighbors, making it an expensive procedure expecially for high dimensional data. Furthermore, it has large memory requirements, since it needs to store the entire data set. To reduce memory requirements, [Low95] implements a process for thinning data in regions where class labels are uniform. Clearly, the effectiveness of this technique depends on the distribution of data, and, in general, the memory requirement will still be much larger than the space utilization of DenClass, which only retains the estimated density function.

11.8 Summary

We have proposed a locally adaptive technique to address the problem of setting the bandwidth parameters optimally for kernel density estimation. We have also shown how to apply our technique to efficiently solve range query approximation, classification and clustering problems for very large data sets.

Our technique manifests a robust and competitive behavior across all the data sets we have considered in our experiments. Moreover, it has the advantage of being simple and can be implemented efficiently in only two data passes.

Chapter 12

Conclusions

This chapter summarizes the dissertation, discusses its contributions, and suggests directions for feature research.

12.1 Summary

We have presented a number of novel techniques to address data exploration tasks such as classification, clustering, and range query approximation. All methods design adaptive metrics or parameter estimates that are local in input space. Both accuracy and efficiency issues have been discussed and addressed.

Pattern classification faces a difficult challenge in finite settings and high dimensional spaces due to the curse of dimensionality. Nearest neighbor methods are especially sensitive to this problem. The need for large neighborhoods in high dimensional spaces is the cause of highly biased estimates. Due to the local nature of feature relevance, any chosen fixed metric violates the

assumption of locally constant class posterior probabilities, and therefore fails in making correct predictions in different regions of the input space. In order to achieve accurate predictions, it becomes crucial to be able to estimate the different degrees of relevance that input features may have in various locations of the feature space.

In this dissertation we have discussed previous work in the literature on flexible metric computation, and introduced novel approaches to computing local feature relevance for nearest neighbor methods that overcome the limitations of previous techniques. The first new approach we have introduced is ADAMENN. The ADAMENN technique uses the Chi-squared distance in order to estimate to which extent each dimension can be relied on to predict class posterior probabilities. The resulting flexible metric produces neighborhoods that are elongated along less relevant features and constricted along most influential ones. As a result, the class conditional probabilities are more homogeneous in the modified neighborhoods. In our experimental evaluation we have shown that ADAMENN exhibits large accuracy improvements with respect to many competitor classifiers. The results obtained strengthen the theoretical properties of our technique.

The dissertation proceeds by discussing the limitations of lazy learning approaches concerned with scalability and efficiency issues. The lazy learning approach employed by ADAMENN requires a considerable amount of on-line computation, which makes it difficult for such technique to scale up to large data sets. To address this issue a new locally flexible metric technique (LFM-SVM) is presented. Although still founded on a query based weighting mechanism, LFM-SVM computes off-line the information relevant to define local weights. This process is guided by the solution provided by a support vector machine. The efficacy of the method is demonstrated through experimental results.

The phenomenon of the curse of dimensionality is not confined to classification. It affects any estimation process in a high dimensional feature space with finite samples. Thus, clustering suffers from the same problem. It is not meaningful to look for clusters in high dimensional spaces as the average density of points anywhere in input space is likely to be low. The dissertation introduces a new algorithm (GenProClus) that discovers clusters in subspaces spanned by different combinations of dimensions via local weightings of features. GenProClus represents an attempt to dodge the curse of dimensionality for clustering problems. The feasibility of our technique is demonstrated through experimental results.

The dissertation concludes by discussing the problem of setting the bandwidth parameters for kernel density estimation. Current work on this problem in statistics solve only the one dimensional case satisfactorily. A new locally adaptive technique (AdaBand) is introduced. The method does not assume independence among the attributes, and exploit possible local correlations among them. It is efficient and can be performed in only two data passes. The resulting density estimation provides a compact representation of the data set, suited to efficiently solve data exploration tasks, such as classification, clustering and range query approximation. The efficiency and accuracy of our techniques are validated in an experimental evaluation.

12.2 Contributions

Listed below are the original contributions of this dissertation.

1. *ADAMENN algorithm*: A novel approach to computing local feature relevance for pattern classification. It uses the Chi-squared distance to design a flexible metric that approximates

the theoretical infinite sample risk at the query point. No assumption is made on the probability distribution of data. It is formally shown that the measure of feature relevance derived by ADAMENN reduces the overall mean-squared estimation error. The technique overcomes limitations of other adaptive methods that employ a greedy strategy, or base the derivation of the metric on discriminant analysis. The theoretical properties of ADAMENN are corroborated by the experimental evaluation.

2. *LFM-SVM algorithm*: A new local flexible metric technique based on support vector machines. This method overcomes the limitations of lazy learning approaches concerned with scalability and efficiency issues. LFM-SVM benefits from the generally sparse solution given by SVMs, and efficiently set the input parameters according to principled guidelines. It is shown that the weighting scheme performed by the LFM-SVM algorithm increases the margin of the solution provided in input by the SVM. It is interesting to note that the LFM-SVM technique can be seen as an attempt to enhance the locality of SVMs. In fact, its transformation metric depends on the orientation of the SVM boundary locally at the query, and on the distance of the query from the closest support vector.

3. *GenProClus algorithm*: A novel algorithm that computes intra-cluster adaptive metrics for clustering. This algorithm represents an attempt to dodge the curse of dimensionality for clustering. The output of the algorithm is twofold. It provides a partition of the data, so that the points in each set of the partition constitute a cluster. In addition, it provides information to what features are relevant for each partition. It is shown that the algorithm converges to a local minimum of the associated error function. The gain in performance achieved with GenProClus with respect to K-means is experimentally demonstrated.

4. *AdaBand algorithm*: A new locally adaptive technique to set the bandwidth parameters for kernel density estimation. This approach can serve as an efficient approximation of nearest neighbor methods. It is shown how this algorithm can be used to efficiently solve classification, clustering, and range query approximation problems. The feasibility of the approach is demonstrated through an experimental evaluation.

12.3 Future Research

There are many possible ways to extend the research described in this dissertation. Here we highlight relevant directions for future work.

We have considered estimating feature relevance along each individual dimension, one at the time, both for classification and clustering. A potential extension is to consider additional derived variables for local relevance estimate, thereby contributing to the distance calculation. The challenge is to be able to have a mechanism that computes such informative derived features efficiently.

The local nature of adaptation performed in ADAMENN makes the idea of considering derived variables particularly attractive. Derived variables can be customized to a specific query point where prediction is to be made, thereby defining different derived variables for different query points. We have discussed the possibility of defining derived variables that are linear combinations of the input features (Section 4.7).

Recent developments on kernel-based methods suggest a general framework for a potential extension of the locally adaptive techniques developed in this dissertation. Kernel PCA [SSM98]

is a non-linear feature extractor that has been proven powerful as a preprocessing step for classification. By the use of suitable *non-linear* features, it can extract patterns from data that cannot be detected by linear PCA. Kernel methods have also been introduced for clustering. Spectral kernel methods [NJW01, BN01] construct the kernel matrix of distances, and make use of the matrix's eigenvectors in order to partition points in clusters.

Following this line of research, the idea of *kernelizing* ADAMENN and GenProClus becomes intriguing. It could allow, by the use of suitable non-linear features, the computation of locally adaptive neighborhoods with arbitrary orientations and shapes in input space.

Bibliography

[APW+99] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, "Fast Algorithms for Projected Clustering", *ACM-SIGMOD Intl. Conference on Management of Data*, 1999.

[AGG+98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications", *ACM-SIGMOD Intl. Conference on Management of Data*, 1998.

[Aha97] D. Aha, "Lazy Learning", *Artificial Intelligence Review*, **11**:1-5, 1997.

[AW99] S. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions", *Neural Networks*, **12**:783-789, 1999.

[AH96] P. Arabie and L. J. Hubert, "An Overview of Combinatorial Data Analysis", *Clustering and Classification*, P. Arabie, L. J. Hubert, and G. D. Soete (editors), World Scientific Pub., 1996.

[AMS97] C. Atkeson, A. W. Moore, and S. Schaal, "Locally Weighted Learning", *Artificial Intelligence Review*, **11**:11-73, 1997.

[BN01] M. Belkin and P. Niyogi, "Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering", *Advances in Neural Information Processing Systems*, 2001.

[Bel61] R. E. Bellman, *Adaptive Control Processes*, Princeton Univ. Press, 1961.

[BFG99] K. P. Bennett, U. Fayyad, and D. Geiger, "Density-Based Indexing for Approximate Nearest-Neighbor Queries", *Intl. Conference on Knowledge Discovery and Data Mining*, 1999.

[Bis95] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

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

[BFR98] P. S. Bradley, U. Fayyad, and C. Reina, "Scaling Clustering Algorithms to Large Datasets", *Intl. Conference on Knowledge Discovery and Data Mining*, 1998.

[Bre96] L. Breiman, "Bagging Predictors", *Machine Learning*, **24**:123-140, 1996.

[Bre99] L. Breiman, "Prediction Games and Arcing Algorithms", *Neural Computation*, **11**:1493-1517, 1999.

[BFO+84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, 1984.

[BMP77] L. Breiman, W. Meisel, and E. Purcell, "Variable Kernel Estimates of Multivariate Densities", *Technometrics*, **19**:135-144.

[BGL+00] M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler, "Knowledge-based Analysis of Microarray Gene Expressions Data Using Support Vector Machines", *Proceedings of the National Academy of Science*, **97**(1):262-267, 2000.

[Bur98] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", *Data Mining and Knowledge Discovery*, **2**(2), 121-167, 1998.

[CP00] G. Cauwenberghs and T. Poggio, "Incremental and Decremental Support Vector Machine Learning", *Advances in Neural Information Processing Systems 13*, MIT Press, 2000.

[CGR+00] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim, "Approximate Query Processing Using Wavelets", *Intl. Conference on Very Large Data Bases*, 2000.

[CM00] K. Chakrabarti and S. Mehrotra, "Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces", *Intl. Conference on Very Large Data Bases*, 2000.

[CS96] P. Cheeseman and J. Stutz, "Bayesian Classification (Autoclass): Theory and Results", *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (editors), AAAI/MIT-Press, 1996.

[CD88] W. S. Cleveland and S. J. Devlin, "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting", *J. Amer. Statist. Assoc.*, **83**:596-610, 1988.

[Cov68] T. M. Cover, "Rates of convergence for nearest neighbor decision procedures", *Hawaii International Conference on System Sciences*, 413-415, 1968.

[CH67] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification", *IEEE Trans. on Information Theory*, **13**(1):21-27, 1967.

[Cre93] N. A. C. Cressie, *Statistics for Spatial Data*, Wiley & Sons, 1993.

[CT01] N. Cristianini and J. Shawe-Taylor *An introduction to Support Vector Machines*, Cambridge University Press, 2001.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm", *Journal of the Royal Statistical Society*, Series B, **39**(1): 1-38, 1977.

[DG01a] C. Domeniconi and D. Gunopulos, "Incremental Support Vector Machine Construction", *First IEEE International Conference on Data Mining*, 2001.

[DG01b] C. Domeniconi and D. Gunopulos, "Adaptive Nearest Neighbor Classification using Support Vector Machines", *Advances in Neural Information Processing Systems 14*, MIT Press, 2001.

[DG01c] C. Domeniconi and D. Gunopulos, "An Efficient Approach for Approximating Multi-dimensional Range Queries and Nearest Neighbor Classification in Large Datasets", *International Conference on Machine Learning*, 2001.

[DG02] C. Domeniconi and D. Gunopulos, "Efficient Local Flexible Nearest Neighbor Classification", *Second SIAM Intl. Conference on Data Mining*, 2002.

[DPG00a] C. Domeniconi, J. Peng, and D. Gunopulos, "Adaptive Metric Nearest Neighbor Classification", *IEEE International Conference on Computer Vision and Pattern Recognition*, 2000.

[DPG00b] C. Domeniconi, J. Peng, and D. Gunopulos, "An Adaptive Metric Machine for Pattern Classification" *Advances in Neural Information Processing Systems 13*, MIT Press, 2000.

[DPG02] C. Domeniconi, J. Peng, and D. Gunopulos, "Locally Adaptive Metric Nearest Neighbor Classification", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **24**(11), 2002 (in print).

[DPV+02] C. Domeniconi, C. S. Perng, R. Vilalta, and S. Ma, "A Classification Approach for Prediction of Target Events in Temporal Sequences", *Sixth European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2002.

[DP96] P. Domingos and M. Pazzani, "Beyond independence: Conditions for the optimality of the simple Bayesian classifier", *Thirteenth International Conference on Machine Learning*, 1996.

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

[DB00] J. G. Dy and C. E. Brodley, "Feature Subset Selection and Order Identification for unsupervised Learning", *International Conference on Machine Learning*, 2000.

[EKX95] M. Ester, H. P. Kriegel, and X. Xu, "A Database Interface for Clustering in Large Spatial Databases", *First International Conference on Knowledge Discovery and Data Mining*, 1995.

[FS96] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm", *Thirteenth International Conference on Machine Learning*, 1996.

[FS95] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *European Conference on Computational Learning Theory*, 1995.

[Fri94] J. H. Friedman "Flexible Metric Nearest Neighbor Classification", Tech. Report, Dept. of Statistics, Stanford University, 1994.

[FF99] J. H. Friedman and N. I. Fisher, "Bump Hunting in High-Dimensional Data", *Statistics and Computing*, **9**(2): 123-143, 1999.

[Fuk90] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.

[Ger82] A. Gersho, "On the structure of vector quantizers", *IEEE Transactions on Information Theory*, **28**(2):157-166, 1982.

[GG91] A. Gersho and R. M. Gray, *Vector Quantization and signal compression*, Kluwer Academic Publishers, 1991.

[GH96] Z. Ghahramani and G. E. Hinton, "The EM Algorithm for Mixtures of Factor Analyzers" Technical Report CRG-TR-96-1, Department of Computer Science, University of Toronto, 1996.

[GKT+00] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi, "Approximating Multi-Dimensional Aggregate Range Queries Over Real Attributes", *19th ACM-SIGMOD Intl. Conference on Management of Data*, 2000.

[HS92] P. J. Haas, and A. N. Swami, "Sequential Sampling Procedures for Query Size Estimation", *ACM-SIGMOD Intl. Conference on Management of Data*, 1992.

[HT96a] T. Hastie and R. Tibshirani, "Discriminant Adaptive Nearest Neighbor Classification", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **18**(6):607-615, 1996.

[HT96b] T. Hastie and R. Tibshirani, "Discriminant Analysis by Gaussian Mixtures", *J. Royal Statistical Society*, B., **58**: 155-176, 1996.

[HK98] A. Hinneburg and D. A. Keim, "An Efficient Approach to Clustering in Large Multimedia Databases with Noise", *Intl. Conference on Knowledge Discovery and Data Mining*, 1998.

[Ho98] T. K. Ho, "Nearest Neighbors in Random Subspaces", *Lecture Notes in Computer Science: Advances in Pattern Recognition*, 640-648, 1998.

[IP99] Y. Ioannidis, and V. Poosala, "Histogram-Based Approximation of Set-Valued Query-Answers", *Intl. Conference on Very Large Data Bases*, 1999.

[JH98] T. S. Jaakkola, D. Haussler, "Exploiting generative models in discriminative classifiers", *Advances in Neural Information Processing Systems 11*, MIT-Press, 1998.

[Joa98] T. Joachims, "Text Categorization with Support Vector Machines", *European Conference on Machine Learning*, 1998.

[Joa99] T. Joachims, "Making Large-Scale SVM Learning Practical", *Advances in Kernel Methods-Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola (editors), MIT-Press, 1999.

[Joa02] T. Joachims, *Learning to Classify Text using Support Vector Machines: Methods, Theory, and Algorithms*, Kluwer Academic Publishers, 2002.

[Jor95] M. I. Jordan, "Why the Logistic Function? A Tutorial Discussion on Probabilities and Neural Networks", *Computational Cognitive Science Technical Report 9503*, Massachusetts Institute of Technology, August 1995.

[KCM+01] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani, "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases", *ACM-SIGMOD Intl. Conference on Management of Data*, 2001.

[LJB+95] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik, "Comparison of Learning Algorithms for Handwritten Digit Recognition", *International Conference on Artificial Neural Networks*, 1995.

[Low95] D. G. Lowe, "Similarity Metric Learning for a Variable-Kernel Classifier", *Neural Computation*, **7**(1):72-85, 1995.

[MRL98] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate Medians and other Quantiles in One Pass and with Limited Memory", *ACM-SIGMOD Intl. Conference on Management of Data*, 1998.

[Mcl92] G. J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*, New York: Wiley, 1992.

[Mei72] W. S. Meisel, *Computer-Oriented Approaches to Pattern Recognition*, Academic Press, New York and London, 1972.

[MS83] R. S. Michalski and R. E. Stepp, "Learning from Observation: Conceptual Clustering", *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (editors), **1**:331-363, Morgan Kaufmann, 1983.

[Mit97] T. M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[MMP00] P. Mitra, C. A. Murthy, and S. K. Pal, "Data Condensation in Large Databases by Incremental Learning with Support Vector Machines", *International Conference on Pattern Recognition*, 2000.

[MD89] J. Moody and C. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units", *Neural Computation*, **1**(2):281-294, 1989.

[MH90] J. P. Myles and D. J. Hand, "The Multi-Class Metric Problem in Nearest Neighbor Discrimination Rules", *Pattern Recognition*, **23**:1291-1297, 1990.

[NH94] R. T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining", *Intl. Conference on Very Large Data Bases*, 1994.

[NJW01] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On Spectral Clustering: Analysis and an Algorithm", Advances in Neural Information Processing Systems, 2001.

[OFG97] E. Osuna, R. Freund, and F. Girosi, "Training Support Vector Machines: An Application to Face Recognition", *International Conference on Computer Vision and Pattern Recognition*, 1997.

[PT92] B. V. Park and B. A. Turlach, "Practical Performance of Several Data Driven Bandwidth Selectors", *Computational Statistics*, 7: 251-270, 1992.

[Pla99] J. C. Platt, "Fast Training of Support Vector Machines using Sequential Minimal Optimization", *Advances in Kernel Methods*, B. Schölkopf, C. J. C. Burges, and A. Smola (editors), MIT-Press, 1999.

[PG90] T. Poggio and F. Girosi, "Networks for Approximation and Learning", *Proceedings of the IEEE*, **78**(9), 1990.

[PV98] M. Pontil and A. Verri, "Object Recognition with Support Vector Machines", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **20**:637-646,1998.

[PI97] V. Poosala and Y. E. Ioannidis, "Selectivity Estimation Without the Attribute Value Independence Assumption", *Intl. Conference on Very Large Data Bases*, 1997.

[Qui86] J. R. Quinlan, "Induction of Decision Trees", *Machine Learning*, **1**:81-106, 1986.

[Qui93] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan-Kaufmann Publishers, Inc., 1993.

[Qui96] J. R. Quinlan, "Bagging, boosting and C4.5", *Fourteenth National Conference on Artificial Intelligence*, 1996.

[Sai99] S. R. Sain, "Multivariate Locally Adaptive Density Estimation", Technical Report, Department of Statistical Science, Southern Methodist University, 1999.

[Sal91] S. Salzberg, "A Nearest Hyperrectangle Learning Method", *Machine Learning*, **6**:251-276, 1991.

[SFB+97] R. Schapire, Y. Freund, P. Bartlett, and W. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods", *14th International Conference on Machine Learning*, 1997.

[SS02] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*, MIT-Press, 2002

[SSM98] B. Schölkopf, A. Smola, and K. R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem", *Neural Computation*, **10**:1299-1319, 1998.

[Sco92] D. Scott, *Multivariate Density Estimation: Theory, Practice and Visualization*., Wiley & Sons, 1992.

[SFB99] J. Shanmugasundaram, U. Fayyad, and P. Bradley, "Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions", *Intl. Conference on Knowledge Discovery and Data Mining*, 1999.

[SF81] R. D. Short and K. Fukunaga, "Optimal Distance Measure for Nearest Neighbor Classification," *IEEE Transactions on Information Theory*, **27**:622-627, 1981.

[Sto77] C. J. Stone, "Nonparametric regression and its applications (with discussion)", *Ann. Statist.* **5**(595), 1977.

[SLS99] N. A. Syed, H. Liu, and K. K. Sung, "Incremental Learning with Support Vector Machines", *International Joint Conference on Artificial Intelligence*, 1999.

[TB99] M. E. Tipping and C. M. Bishop, "Mixtures of Principal Component Analyzers", *Neural Computation*, **11**(2): 443-482, 1999.

[TS92] G. R. Terrell and D. W. Scott, "Variable Kernel Density Estimation", *The Annals of Statistics*, **20**: 1236-1265.

[TCL98] A. Thomasian, V. Castello and C. S. Li, "Clustering and Singular Value Decomposition for Approximate Indexing in High Dimensional Spaces", *CIKM*, 1998.

[Vap98] V. N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.

[Vap99] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Second Edition, Springer-Verlag, 1999.

[VCC99] K. Veropoulos, C. Campbell, and N. Cristianini, "Controlling the Sensitivity of Support Vector Machines", *IJCAI Workshop on Support Vector Machines*, 1999.

[VWI98] J. S. Vitter, M. Wang, and B. R. Iyer, "Data Cube Approximation and Histograms via Wavelets", *ACM CIKM Intl. Conference on Information and Knowledge Management*, 1998.

[WJ95] M. P. Wand and M. C. Jones, "Kernel Smoothing", *Monographs on Statistics and Applied Probability*, Chapman & Hall, 1995.

[WSB98] R. Weber, H. J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity Search Methods in High-Dimensional Spaces", *Intl. Conference on Very Large Data Bases*, 1998.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases", *ACM-SIGMOD Intl. Conference on Management of Data*, 1996.