Statistical Pattern Recognition: A Review

Anil K. Jain (Fellow, IEEE)*

Robert P.W. Duin† and Jianchang Mao (Senior Member, IEEE)‡

November 1, 1999

Abstract

The primary goal of pattern recognition is supervised or unsupervised classification. Among the various frameworks in which pattern recognition has been traditionally formulated, the statistical approach has been most intensively studied and used in practice. More recently, neural network techniques and methods imported from statistical learning theory have been deserving increasing attention. The design of a recognition system requires careful attention to the following issues: definition of pattern classes, sensing environment, pattern representation, feature extraction and selection, cluster analysis, classifier design and learning, selection of training and test samples, and performance evaluation. In spite of almost fifty years of research and development in this field, the general problem of recognizing complex patterns with arbitrary orientation, location, and scale remains unsolved. New

*Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA

†Pattern Recognition Group, Department of Applied Physics, Delft University of Technology, The Netherlands

‡IBM Almaden Research Center, San Jose, CA 95120, USA

and emerging applications, such as data mining, web searching, retrieval of multimedia data, face recognition and cursive handwriting recognition, require robust and efficient pattern recognition techniques. The objective of this review paper is to summarize and compare some of the well-known methods used in various stages of a pattern recognition system and identify research topics and applications which are at the forefront of this exciting and challenging field.

Keywords: Statistical pattern recognition, classification, clustering, feature extraction, feature selection, error estimation, classifier combination, neural networks.

1 Introduction

By the time they are five years old, most children can recognize digits and letters. Small characters, large characters, handwritten, machine printed, or rotated—all are easily recognized by the young. The characters may be written on a cluttered background, on crumpled paper or may even be partially occluded. We take this ability for granted until we face the task of teaching a machine how to do the same. Pattern recognition is the study of how machines can observe the environment, learn to distinguish patterns of interest from their background, and make sound and reasonable decisions about the categories of the patterns. In spite of almost fifty years of research, design of a general-purpose machine pattern recognizer remains an elusive goal.

The best pattern recognizers in most instances are humans, yet we do not understand how humans recognize patterns. Ross [140] emphasizes the work of Nobel Laureate Herbert Simon whose central finding was that pattern recognition is critical in most human decision making tasks: "The more relevant patterns at your disposal, the better your decisions will be. This is hopeful news to proponents of artificial intelligence, since computers can surely be taught to recognize patterns. Indeed, successful computer programs that help banks store credit applicants, help doctors diagnose disease and help pilots land airplanes depend in some way on pattern recognition. ... We need to pay much more explicit attention to teaching pattern recognition." Our goal here is to introduce pattern recognition as the best possible way of utilizing available sensors, processors, and domain knowledge to make decisions automatically.

1.1 What is Pattern Recognition?

Automatic (machine) recognition, description, classification, and grouping of patterns are important problems in a variety of engineering and scientific disciplines such as biology, psychology, medicine, marketing, computer vision, artificial intelligence, and remote sensing. But what is a pattern? Watanabe [163] defines a pattern "as opposite of a chaos; it is an entity, vaguely defined, that could be given a name." For example, a pattern could be a fingerprint image, a handwritten cursive word, a human face, or a speech signal. Given a pattern, its recognition/classification may consist of one of the following two tasks [163]: (i) supervised classification (e.g., discriminant analysis) in which the input pattern is identified as a member of a predefined class, (ii) unsupervised classification (e.g., clustering) in which the pattern is assigned to a hitherto unknown class. Note that the recognition problem here is being posed as a classification or categorization task, where the classes are either defined by the system designer (in supervised classification) or are learned based on the similarity of patterns (in unsupervised classification).

Interest in the area of pattern recognition has been renewed recently due to emerging applications which are not only challenging but also computationally more demanding (see Table 1). These applications include data mining (identifying a "pattern", e.g., cor-

Table 1: Examples of Pattern Recognition Applications

Problem Domain	Application	Input Pattern	Pattern Classes
Bioinformatics	Sequence Analysis	DNA/Protein sequence	Known types of genes/ patterns
Data mining	Searching for meaningful patterns	Points in multi-dimensional space	Compact and well-separated clusters
Document classification	Internet search	Text document	Semantic categories (e.g., business, sports, etc.)
Document image analysis	Reading machine for the blind	Document image	Alphanumeric characters, words
Industrial automation	Printed circuit board inspection	Intensity or range image	Defective / non-defective nature of product
Multimedia database retrieval	Internet search	Video clip	Video genres (e.g., action, dialogue, etc.)
Biometric recognition	Personal identification	Face, iris, fingerprint	Authorized users for access control
Remote sensing	Forecasting crop yield	Multispectral image	Land use categories, growth pattern of crops
Speech recognition	Telephone directory enquiry without operator assistance	Speech waveform	Spoken words

relation, or an outlier in millions of multi-dimensional patterns), document classification (efficiently searching text documents), financial forecasting, organization and retrieval of multimedia databases, and biometrics (personal identification based on various physical attributes such as faces and fingerprints). Picard [125] has identified a novel application of pattern recognition, called affective computing which will give a computer the ability to recognize and express emotions, to respond intelligently to human emotion, and to employ mechanisms of emotion that contribute to rational decision making. A common characteristic of a number of these applications is that the available features (typically, in the thousands) are not usually suggested by domain experts, but must be extracted and optimized by data-driven procedures.

The rapidly growing and available computing power, while enabling faster processing of huge data sets, has also facilitated the use of elaborate and diverse methods for data analysis and classification. At the same time, demands on automatic pattern recognition systems are rising enormously due to the availability of large databases and stringent performance requirements (speed, accuracy and cost). In many of the emerging applications, it is clear that no single approach for classification is "optimal" and that multiple methods and approaches have to be used. Consequently, combining several sensing modalities and classifiers is now a commonly used practice in pattern recognition.

The design of a pattern recognition system essentially involves the following three aspects: (i) data acquisition and preprocessing, (ii) data representation, and (iii) decision making. The problem domain dictates the choice of sensor(s), preprocessing technique, representation scheme, and the decision making model. It is generally agreed that a well-defined and sufficiently constrained recognition problem (small intra-class variations and large inter-class variations) will lead to a compact pattern representation and a simple decision making strategy. Learning from a set of examples (training set) is an important and desired attribute of most pattern recognition systems. The four best known approaches for pattern recognition are: (i) template matching, (ii) statistical classification, (iii) syntactic or structural matching, and (iv) neural networks. These models are not necessarily independent and sometimes the same pattern recognition method exists with different interpretations. Attempts have been made to design hybrid systems involving multiple models [57]. A brief description and comparison of these approaches is given below and summarized in Table 2.

Table 2: Pattern recognition models

Approach	Representation	Recognition function	Typical criterion
Template matching	Samples, pixels, curves	Correlation, distance measure	Classification error
Statistical	Features	Discriminant function	Classification error
Syntactic or structural	Primitives	Rules, grammar	Acceptance error
Neural networks	Samples, pixels, features	Network function	Mean square error

1.2 Template Matching

One of the simplest and earliest approaches to pattern recognition is based on template matching. Matching is a generic operation in pattern recognition which is used to determine the similarity between two entities (points, curves or shapes) of the same type. In template matching, a template (typically, a $2D$ shape) or a prototype of the pattern to be recognized is available. The pattern to be recognized is matched against the stored template while taking into account all allowable pose (translation and rotation) and scale changes. The similarity measure, often a correlation, may be optimized based on the available training set. Often, the template itself is learned from the training set. Template matching is computationally demanding, but the availability of faster processors has now made this approach more feasible. The rigid template matching mentioned above, while effective in some application domains, has a number of disadvantages. For instance, it would fail if the patterns are distorted due to the imaging process, view-point change, or large intra-class variations among the patterns. Deformable template models

[69] or rubber sheet deformations [9] can be used to match patterns when the deformation cannot be easily explained or modeled directly.

1.3 Statistical Approach

In the statistical approach, each pattern is represented in terms of d features or measurements and is viewed as a point in a d-dimensional space. The goal is to choose those features that allow pattern vectors belonging to different categories to occupy compact and disjoint regions in a d-dimensional feature space. The effectiveness of the representation space (feature set) is determined by how well patterns from different classes can be separated. Given a set of training patterns from each class, the objective is to establish decision boundaries in the feature space which separate patterns belonging to different classes. In the statistical decision theoretic approach, the decision boundaries are determined by the probability distributions of the patterns belonging to each class, which must either be specified or learned [41], [44].

One can also take a discriminant analysis-based approach to classification: first a parametric form of the decision boundary (e.g., linear or quadratic) is specified; then the "best" decision boundary of the specified form is found based on the classification of training patterns. Such boundaries can be constructed using, for example, a mean squared error criterion. The direct boundary construction approaches are supported by Vapnik's philosophy [162]: "If you possess a restricted amount of information for solving some problem, try to solve the problem directly and never solve a more general problem as an intermediate step. It is possible that the available information is sufficient for a direct solution but is insufficient for solving a more general intermediate problem."

1.4 Syntactic Approach

In many recognition problems involving complex patterns, it is more appropriate to adopt a hierarchical perspective where a pattern is viewed as being composed of simple sub-patterns which are themselves built from yet simpler sub-patterns [56, 121]. The simplest/elementary sub-patterns to be recognized are called *primitives* and the given complex pattern is represented in terms of the interrelationships between these primitives. In syntactic pattern recognition, a formal analogy is drawn between the structure of patterns and the syntax of a language. The patterns are viewed as sentences belonging to a language, primitives are viewed as the alphabet of the language, and the sentences are generated according to a grammar. Thus, a large collection of complex patterns can be described by a small number of primitives and grammatical rules. The grammar for each pattern class must be inferred from the available training samples.

Structural pattern recognition is intuitively appealing because, in addition to classification, this approach also provides a description of how the given pattern is constructed from the primitives. This paradigm has been used in situations where the patterns have a definite structure which can be captured in terms of a set of rules, such as EKG waveforms, textured images, and shape analysis of contours [56]. The implementation of a syntactic approach, however, leads to many difficulties which primarily have to do with the segmentation of noisy patterns (to detect the primitives) and the inference of the grammar from training data. Fu [56] introduced the notion of attributed grammars which unifies syntactic and statistical pattern recognition. The syntactic approach may yield a combinatorial explosion of possibilities to be investigated, demanding large training sets and very large computational efforts [122].

1.5 Neural Networks

Neural networks can be viewed as massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections. Neural network models attempt to use some organizational principles (such as learning, generalization, adaptivity, fault tolerance and distributed representation and computation) in a network of weighted directed graphs in which the nodes are artificial neurons and directed edges (with weights) are connections between neuron outputs and neuron inputs. The main differences between neural networks and the other approaches to pattern recognition are that these networks have the ability to learn complex non-linear input-output relationships, and use sequential training procedures. Moreover, they have the general characteristic of adapting themselves to the data.

The most commonly used family of neural networks for pattern classification tasks [83] is the feed-forward network, which includes multilayer perceptron and Radial-Basis Function (RBF) networks. These networks are organized into layers and have unidirectional connections between the layers. Another popular network is the Self-Organizing Map (SOM), or Kohonen-Network [92], which is mainly used for data clustering and feature mapping. The learning process involves updating network architecture and connection weights so that a network can efficiently perform a specific classification/clustering task. The increasing popularity of neural network models to solve pattern recognition problems has been primarily due to their seemingly low dependence on domain-specific knowledge (relative to model-based and rule-based approaches) and due to the availability of efficient learning algorithms for practitioners to use.

Neural networks provide a new suite of nonlinear algorithms for feature extraction (using hidden layers) and classification (e.g., multilayer perceptrons). In addition, existing feature extraction and classification algorithms can also be mapped on neural network architectures for efficient (hardware) implementation. In spite of the seemingly different underlying principles, most of the well known neural network models are implicitly equivalent or similar to classical statistical pattern recognition methods (see Table 3). Ripley [136] and Anderson et al. [5] also discuss this relationship between neural networks and statistical pattern recognition. Anderson et al. point out that "neural networks are statistics for amateurs . . . Most NNs conceal the statistics from the user." Despite these similarities, neural networks do offer several advantages such as, unified approaches for feature extraction and classification and flexible procedures for finding good, moderately nonlinear solutions.

Table 3: Links between statistical and neural network methods

Statistical Pattern Recognition	Artificial Neural Networks
Linear Discriminant Function	Perceptron
Principal Component Analysis	Auto-Associative Network, and various PCA networks
A Posteriori Probability Estimation	Multilayer Perceptron
Nonlinear Discriminant Analysis	Multilayer Perceptron
Parzen Window Density-based Classifier	Radial Basis Function Network
Edited K-NN Rule	Kohonen's LVQ

1.6 Scope and Organization

In the remainder of this paper we will primarily review statistical methods for pattern representation and classification, emphasizing recent developments. Whenever appropriate, we will also discuss closely related algorithms from the neural networks literature. We omit the whole body of literature on fuzzy classification and fuzzy clustering which are in our opinion beyond the scope of this paper. Interested readers can refer to the well-written books on fuzzy pattern recognition by Bezdek [15] and by Bezdek and Pal [16]. In most of the sections, the various approaches and methods are summarized in tables as an easy and quick reference for the reader. Due to space constraints, we are not able to provide many details and we have to omit some of the approaches and the associated references. Our goal is to emphasize those approaches which have been extensively evaluated and demonstrated to be useful in practical applications, along with the new trends and ideas.

The literature on pattern recognition is vast and scattered in numerous journals in several disciplines (e.g., applied statistics, machine learning, neural networks, and signal and image processing). A quick scan of the table of contents of all the issues of the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, since its first publication in January 1979, reveals that approximately 350 papers deal with pattern recognition. Approximately 300 of these papers covered the statistical approach and can be broadly categorized into the following subtopics: curse of dimensionality (15), dimensionality reduction (50), classifier design (175), classifier combination (10), error estimation (25) and unsupervised classification (50). In addition to the excellent textbooks by Duda and Hart [44][1], Fukunaga [58], Devijver and Kittler [39], Devroye, Gyorfi and Lugosi [41], Bishop [18], Ripley [137], Schurmann [147], and McLachlan [105], we should also point out two excellent survey papers written by Nagy [111] in 1968 and by Kanal [89] in 1974. Nagy described the early roots of pattern recognition, which at that time was shared with researchers in artificial intelligence and perception. A large part of Nagy's paper introduced a number of potential applications of pattern recognition and the interplay between feature definition and the application domain knowledge. He also emphasized

[1]Its second edition by Duda, Hart, and Stork is in press.

the linear classification methods; nonlinear techniques were based on polynomial discriminant functions as well as on potential functions (similar to what are now called the kernel functions). By the time Kanal wrote his survey paper, more than 500 papers and about half-a-dozen books on pattern recognition were already published. Kanal placed less emphasis on applications, but more on modeling and design of pattern recognition systems. The discussion on automatic feature extraction in [89] was based on various distance measures between class-conditional probability density functions and the resulting error bounds. Kanal's review also contained a large section on structural methods and pattern grammars.

In comparison to the state of the pattern recognition field as described by Nagy and Kanal in the sixties and seventies, today a number of commercial pattern recognition systems are available which even individuals can buy for personal use (e.g., machine printed character recognition and isolated spoken word recognition). This has been made possible by various technological developments resulting in the availability of inexpensive sensors and powerful desktop computers. The field of pattern recognition has become so large that in this review we had to skip detailed descriptions of various applications, as well as almost all the procedures which model domain-specific knowledge (e.g., structural pattern recognition, and rule-based systems). The starting point of our review (Section 2) is the basic elements of statistical methods for pattern recognition. It should be apparent that a feature vector is a representation of real world objects; the choice of the representation strongly influences the classification results.

The topics of probabilistic distance measures and error bounds are currently not as important as 20 years ago, since it is very difficult to estimate density functions in high dimensional feature spaces. Instead, the complexity of classification procedures and the resulting accuracy have gained a large interest. The curse of dimensionality (Section 3) as well as the danger of overtraining are some of the consequences of a complex classifier. It is now understood that these problems can, to some extent, be circumvented using regularization, or can even be completely resolved by a proper design of classification procedures. The study of support vector machines (SVMs), discussed in Section 5, has largely contributed to this understanding. In many real world problems, patterns are scattered in high-dimensional (often) nonlinear subspaces. As a consequence, nonlinear procedures and subspace approaches have become popular, both for dimensionality reduction (Section 4) and for building classifiers (Section 5). Neural networks offer powerful tools for these purposes. It is now widely accepted that no single procedure will completely solve a complex classification problem. There are many admissible approaches, each capable of discriminating patterns in certain portions of the feature space. The combination of classifiers has, therefore, become a heavily studied topic (Section 6). Various approaches to estimating the error rate of a classifier are presented in Section 7. The topic of unsupervised classification or clustering is covered in Section 8. Finally, Section 9 identifies the frontiers of pattern recognition.

It is our goal that most parts of the paper can be appreciated by a newcomer to the field of pattern recognition. To this purpose, we have included a number of examples to illustrate the performance of various algorithms. Nevertheless, we realize that, due to space limitations, we have not been able to introduce all the concepts completely. At these places, we have to rely on the background knowledge which may be available only to the more experienced readers.

2 Statistical Pattern Recognition

Statistical pattern recognition has been used successfully to design a number of commercial recognition systems. In *statistical* pattern recognition, a pattern is represented by a set of d features, or attributes, viewed as a d-dimensional feature vector. Well-known concepts from statistical decision theory are utilized to establish decision boundaries between pattern classes. The recognition system is operated in two modes: training (learning) and classification (testing) (see Fig. 1). The role of the preprocessing module is to segment the pattern of interest from the background, remove noise, normalize the pattern, and any other operation which will contribute in defining a compact representation of the pattern. In the training mode, the feature extraction/selection module finds the appropriate features for representing the input patterns and the classifier is trained to partition the feature space. The feedback path allows a designer to optimize the preprocessing and feature extraction/selection strategies. In the classification mode, the trained classifier assigns the input pattern to one of the pattern classes under consideration based on the measured features.

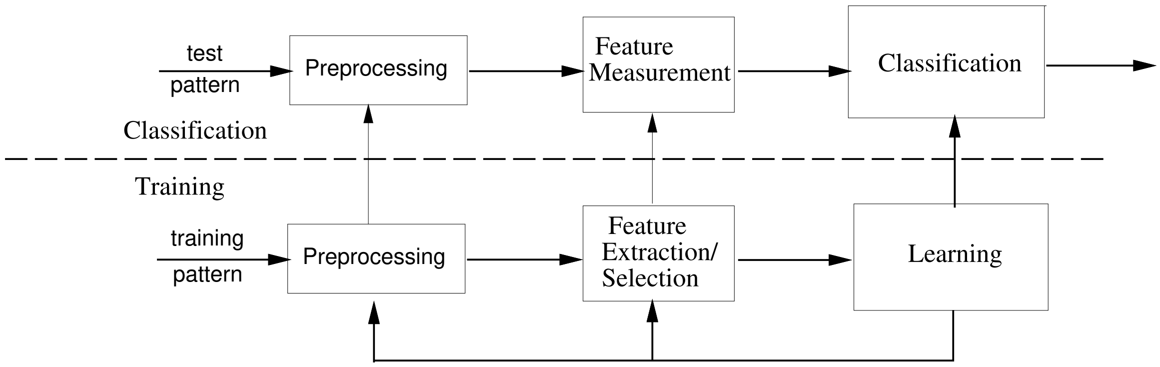


Figure 1: Model for statistical pattern recognition.

The decision making process in statistical pattern recognition can be summarized as follows. A given pattern is to be assigned to one of c categories ω_1, ω_2, ..., ω_c based on a vector of d feature values $\boldsymbol{x} = (x_1, x_2, \cdots, x_d)$. The features are assumed to have a probability density or mass (depending on whether the features are continuous or discrete) function conditioned on the pattern class. Thus, a pattern vector $\boldsymbol{x}$ belonging to class ω_i is viewed as an observation drawn randomly from the class-conditional probability function $p(\boldsymbol{x}|\omega_i)$. A number of well-known decision rules, including the Bayes decision rule, the maximum likelihood rule (which can be viewed as a particular case of the Bayes rule), and the Neyman-Peason rule are available to define the decision boundary. The "optimal" Bayes decision rule for minimizing the risk (expected value of the loss function) can be stated as follows: Assign input pattern $\boldsymbol{x}$ to class ω_i for which the conditional risk

$$R(\omega_i|\boldsymbol{x}) = \sum_{j=1}^{c} L(\omega_i, \omega_j) \cdot P(\omega_j|\boldsymbol{x}) \tag{1}$$

is minimum, where $L(\omega_i, \omega_j)$ is the loss function incurred in deciding ω_i when the true class is ω_j and $P(\omega_j|\boldsymbol{x})$ is the posterior probability [44]. In the case of the 0/1 loss function, as defined in Eq. 2, the conditional risk becomes the conditional probability of misclassification.

$$L(\omega_i, \omega_j) = \begin{cases} 0, & i = j \\ 1, & i \neq j \end{cases} \tag{2}$$

For this choice of loss function, the Bayes decision rule can be simplified as follows (also called the maximum *a posteriori* (MAP) rule): Assign input pattern $\boldsymbol{x}$ to class ω_i if

$$P(\omega_i|\boldsymbol{x}) > P(\omega_j|\boldsymbol{x}) \text{ for all } j \neq i. \tag{3}$$

Various strategies are utilized to design a classifier in statistical pattern recognition, depending on the kind of information available about the class-conditional densities. If all of the class-conditional densities are completely specified, then the optimal Bayes decision rule can be used to design a classifier. However, the class-conditional densities are usually not known in practice and must be learned from the available training patterns. If the form of the class-conditional densities is known (e.g., multivariate Gaussian), but some of the parameters of the densities (e.g., mean vectors and covariance matrices) are unknown, then we have a parametric decision problem. A common strategy for this kind of problem is to replace the unknown parameters in the density functions by their estimated values, resulting in the so-called Bayes plug-in classifier. The optimal Bayesian strategy in this situation requires additional information in the form of a prior distribution on the unknown parameters. If the form of the class-conditional densities is not known, then we operate in an nonparametric mode. In this case, we must either estimate the density function (e.g., Parzen window approach) or directly construct the decision boundary based on the training data (e.g., k-nearest neighbor rule). In fact, the multilayer perceptron can also be viewed as a supervised nonparametric method which constructs a decision boundary.

Another dichotomy in statistical pattern recognition is that of supervised learning (labeled training samples) versus unsupervised learning (unlabeled training samples). The label of a training pattern represents the category to which that pattern belongs. In an unsupervised learning problem, sometimes the number of classes must be learned along with the structure of each class. The various dichotomies that appear in statistical pattern recognition are shown in the tree structure of Fig. 2. As we traverse the tree from top to bottom and left to right, less *information* is available to the system designer and as a result, the difficulty of classification problem increases. In some sense, most of the approaches in statistical pattern recognition (leaf nodes in the tree of Fig. 2) are attempting to implement the Bayes decision rule. The field of cluster analysis essentially deals with decision making problems in the nonparametric and unsupervised learning mode [81]. Further, in cluster analysis the number of categories or clusters may not even be specified; the task is to discover a reasonable categorization of the data (if one exists). Cluster analysis algorithms along with various techniques for visualizing and projecting multi-dimensional data are also referred to as *exploratory data analysis* methods.

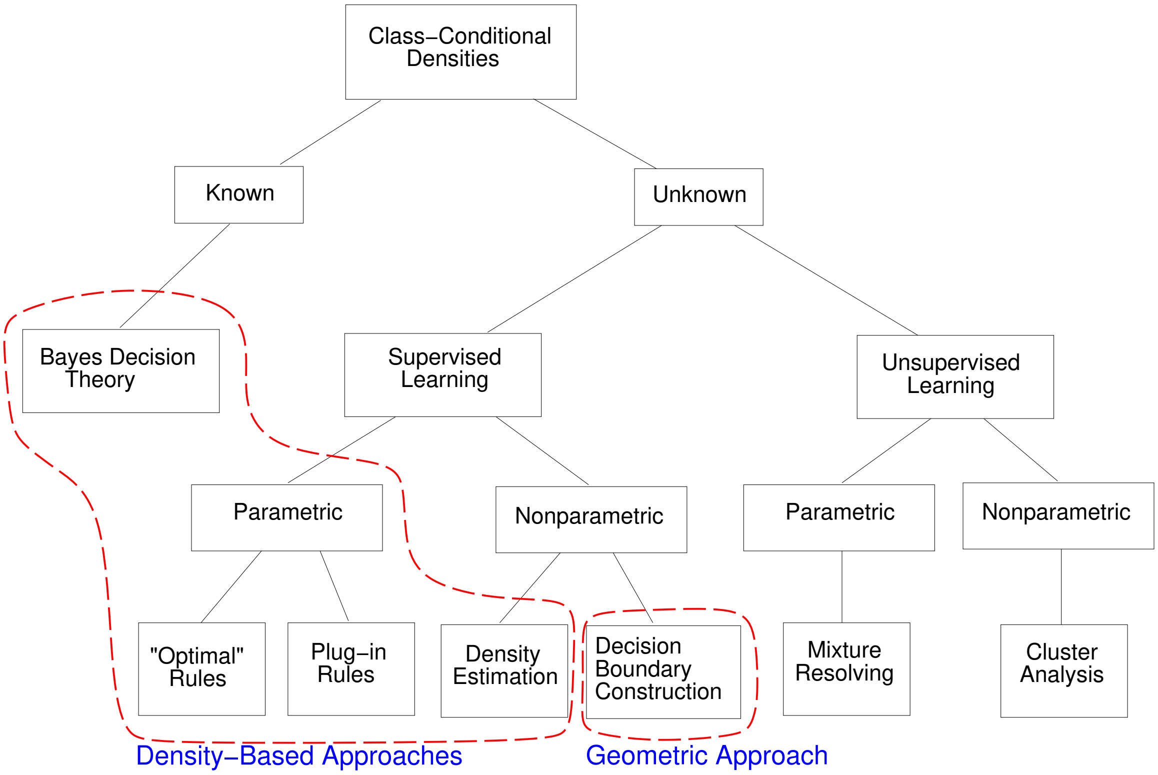


Figure 2: Various approaches in statistical pattern recognition.

Yet another dichotomy in statistical pattern recognition can be based on whether the decision boundaries are obtained directly (geometric approach) or indirectly (probabilistic density-based approach) as shown in Fig. 2. The probabilistic approach requires to estimate density functions first, and then construct the discriminant functions which specify the decision boundaries. On the other hand, the geometric approach often constructs the decision boundaries directly from optimizing certain cost functions. We should point out that under certain assumptions on the density functions, the two approaches are equivalent. We will see examples of each category in Section 5.

No matter which classification or decision rule is used, it must be trained using the available training samples. As a result, the performance of a classifier depends on both the number of available training samples as well as the specific values of the samples. At the same time, the goal of designing a recognition system is to classify future test samples which are likely to be different from the training samples. Therefore, optimizing a classifier to maximize its performance on the training set may not always result in the desired performance on a test set. The generalization ability of a classifier refers to its performance in classifying test patterns which were not used during the training stage. A poor generalization ability of a classifier can be attributed to any one of the following factors: (i) the number of features is too large relative to the number of training samples (curse of dimensionality [80]), (ii) the number of unknown parameters associated with the classifier is large (e.g., polynomial classifiers or a large neural network), and (iii) a classifier is too intensively optimized on the training set (overtrained); this is analogous to the phenomenon of overfitting in regression when there are too many free parameters.

Overtraining has been investigated theoretically for classifiers that minimize the apparent error rate (the error on the training set). The classical studies by Cover [33] and Vapnik [162] on classifier capacity and complexity provide a good understanding of the mechanisms behind overtraining. Complex classifiers (e.g., those having many independent parameters) may have a large capacity, i.e. they are able to represent many dichotomies for a given dataset. A frequently used measure for the capacity is the Vapnik-

Chervonenkis (VC) dimensionality [162]. These results can also be used to prove some interesting properties, for example, the consistency of certain classifiers (see, Devroye et al. [40, 41]). The practical use of the results on classifier complexity was initially limited because the proposed bounds on the required number of (training) samples were too conservative. In the recent development of support vector machines [162], however, these results have proved to be quite useful. The pitfalls of over-adaptation of estimators to the given training set are observed in several stages of a pattern recognition system, such as dimensionality reduction, density estimation, and classifier design. A sound solution is to always use an independent dataset (test set) for evaluation. In order to avoid the necessity of having several independent test sets, estimators are often based on rotated subsets of the data, preserving different parts of the data for optimization and evaluation [166]. Examples are the optimization of the covariance estimates for the Parzen kernel [76] and discriminant analysis [61], and the use of bootstrapping for designing classifiers [48], and for error estimation [82].

Throughout the paper, some of the classification methods will be illustrated by simple experiments on the following three data sets:

Dataset 1: An artificial dataset consisting of two classes with bivariate Gaussian density with the following parameters: $\boldsymbol{m_1} = (1,1)$, $\boldsymbol{m_2} = (2,0)$, $\boldsymbol{\Sigma_1} = \begin{bmatrix} 1 & 0 \\ 0 & 0.25 \end{bmatrix}$ and $\boldsymbol{\Sigma_2} = \begin{bmatrix} 0.8 & 0 \\ 0 & 1 \end{bmatrix}$. The intrinsic overlap between these two densities is 12.5%.

Dataset 2: Iris dataset consists of 150 4-dimensional patterns in three classes (50 patterns each): Iris Setosa, Iris Versicolor and Iris Virginica.

Dataset 3: The digit dataset consists of handwritten numerals ('0'–'9') extracted from a collection of Dutch utility maps. Two hundred patterns per class (for a total of 2,000 patterns) are available in the form of 30×48 binary images. These characters are represented in terms of the following six feature sets: (i) 76 Fourier coefficients of the character shapes; (ii) 216 profile correlations; (iii) 64 Karhunen-Loève coefficients; (iv) 240 pixel averages in 2×3 windows; (v) 47 Zernike moments; (vi) 6 morphological features. Details of this dataset are available in [160]. In our experiments we always used the same subset of 1,000 patterns for testing and various subsets of the remaining 1,000 patterns for training[2]. Throughout this paper, when we refer to "the digit dataset", just the Karhunen-Loeve features (iii) are meant, unless stated otherwise.

3 The Curse of Dimensionality and Peaking Phenomena

The performance of a classifier depends on the interrelationship between sample sizes, number of features, and classifier complexity. A naive table-lookup technique (partitioning the feature space into cells and associating a class label with each cell) requires the number of training data points to be an exponential function of the feature dimension [18]. This phenomenon is termed as "curse of dimensionality", which leads to the "peaking phenomenon" (see discussion below) in classifier design. It is well known that the probability of misclassification of a decision rule does not increase as the number of features increases,

[2]The dataset is available through the UCI Machine Learning Repository (http://www.ics.uci.edu/ mlearn/MLRepository.html)

as long as the class-conditional densities are completely known (or, equivalently, the number of training samples is arbitrarily large and representative of the underlying densities). However, it has been often observed in practice that the added features may actually degrade the performance of a classifier if the number of training samples that are used to design the classifier is small relative to the number of features. This paradoxical behavior is referred to as the peaking phenomenon[3] [80, 131, 132]. A simple explanation for this phenomenon is as follows. The most commonly used parametric classifiers estimate the unknown parameters and plug them in for the true parameters in the class-conditional densities. For a fixed sample size, as the number of features is increased (with a corresponding increase in the number of unknown parameters), the reliability of the parameter estimates decreases. Consequently, the performance of the resulting plug-in classifiers, for a fixed sample size, may degrade with an increase in the number of features.

Trunk [157] provided a simple example to illustrate the curse of dimensionality which we reproduce below. Consider the two-class classification problem with equal prior probabilities, and a d-dimensional multivariate Gaussian distribution with the identity covariance matrix for each class. The mean vectors for the two classes have the following components

$$\boldsymbol{m_1} = (1, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}, \cdots, \frac{1}{\sqrt{d}}) \qquad \text{and} \qquad \boldsymbol{m_2} = (-1, -\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{3}}, \cdots, -\frac{1}{\sqrt{d}}).$$

Note that the features are statistically independent and the discriminating power of the successive features decreases monotonically with the first feature providing the maximum discrimination between the two classes. The only parameter in the densities is the mean vector, $\boldsymbol{m} = \boldsymbol{m_1} = -\boldsymbol{m_2}$.

[3] In the rest of this paper, we do not make distinction between the curse of dimensionality and the peaking phenomenon.

Trunk considered the following two cases:

(i) The mean vector $\boldsymbol{m}$ is known. In this situation, we can use the optimal Bayes decision rule (with a 0/1 loss function) to construct the decision boundary. The probability of error as a function of d can be expressed as:

$$P_e(d) = \int_{\sqrt{\sum_{i=1}^{d}(\frac{1}{i})}}^{\infty} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{1}{2}z^2} dz. \tag{4}$$

It is easy to verify that $\lim_{d\to\infty} P_e(d) = 0$. In other words, we can perfectly discriminate the two classes by arbitrarily increasing the number of features, d.

(ii) The mean vector $\boldsymbol{m}$ is unknown and n labeled training samples are available. Trunk found the maximum likelihood estimate $\hat{\boldsymbol{m}}$ of $\boldsymbol{m}$ and used the plug-in decision rule (substitute $\hat{\boldsymbol{m}}$ for $\boldsymbol{m}$ in the optimal Bayes decision rule). Now the probability of error which is a function of both n and d can be written as:

$$P_e(n, d) = \int_{\theta(d)}^{\infty} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{1}{2}z^2} dz \text{ , where} \tag{5}$$

$$\theta(d) = -\frac{\sum_{i=1}^{d}(\frac{1}{i})}{\sqrt{(1+\frac{1}{n})\sum_{i=1}^{d}(\frac{1}{i}) + \frac{d}{n}}}. \tag{6}$$

Trunk showed that $\lim_{d\to\infty} P_e(n, d) = \frac{1}{2}$, which implies that the probability of error approaches the maximum possible value of 0.5 for this two-class problem. This demonstrates that unlike case (i), we cannot arbitrarily increase the number of features when the parameters of class-conditional densities are estimated from a finite number of training samples. The practical implication of the curse of dimensionality is that a system designer should try to select only a small number of salient features when confronted with a limited training set.

All of the commonly used classifiers, including multilayer feed-forward networks, can suffer from the curse of dimensionality. While an exact relationship between the probability of misclassification, the number of training samples, the number of features and the true parameters of the class-conditional densities is very difficult to establish, some guidelines have been suggested regarding the ratio of the sample size to dimensionality. It is generally accepted that using at least ten times as many training samples per class as the number of features ($n/d > 10$) is a good practice to follow in classifier design [80]. The more complex the classifier, the larger should the ratio of sample size to dimensionality be to avoid the curse of dimensionality.

4 Dimensionality Reduction

There are two main reasons to keep the dimensionality of the pattern representation (i.e., the number of features) as small as possible: measurement cost and classification accuracy. A limited yet salient feature set simplifies both the pattern representation and the classifiers that are built on the selected representation. Consequently, the resulting classifier will be faster and will use less memory. Moreover, as stated earlier, a small number of features can alleviate the curse of dimensionality when the number of training samples is limited. On the other hand, a reduction in the number of features may lead to a loss in the discrimination power and thereby lower the accuracy of the resulting recognition system. Watanabe's *ugly duckling theorem* [163] also supports the need for a careful choice of the features, since it is possible to make two arbitrary patterns *similar* by encoding them with a sufficiently large number of redundant features.

It is important to make a distinction between feature selection and feature extraction. The term feature selection refers to algorithms that select the (hopefully) best subset of the input feature set. Methods that create new features based on transformations or combinations of the original feature set are called feature extraction algorithms. However, the terms feature selection and feature extraction are used interchangeably in the literature. Note that often feature extraction precedes feature selection; first, features are extracted from the sensed data (e.g., using principal component or discriminant analysis) and then some of the extracted features with low discrimination ability are discarded. The choice between feature selection and feature extraction depends on the application domain and the specific training data which is available. Feature selection leads to savings in measurement cost (since some of the features are discarded) and the selected features retain their original physical interpretation. In addition, the retrieved features may be important for understanding the physical process that generates the patterns. Finding features can thereby be a goal on its own. On the other hand, transformed features generated by feature extraction may provide a better discriminative ability than the best subset of given features, but these new features (a linear or a nonlinear combination of given features) may not have a clear physical meaning.

In many situations, it is useful to obtain a two- or three- dimensional projection of the given multivariate data ($n \times d$ pattern matrix) to permit a visual examination of the data. Several graphical techniques also exist for visually observing multivariate data, in which the objective is to exactly depict each pattern as a picture with d degrees of freedom, where d is the given number of features. For example, Chernoff [29] represents each pattern as a cartoon face whose facial characteristics, such as nose length, mouth curvature, and eye size, are made to correspond to individual features. Fig. 3 shows three faces corresponding to the mean vectors of Iris Setosa, Iris Versicolor, and Iris Virginica classes in the Iris data (150 4-dimensional patterns; 50 patterns per class). Note that

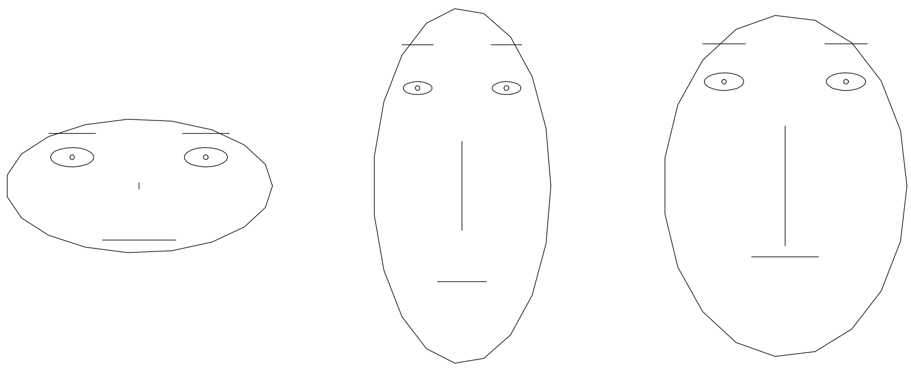


Figure 3: Chernoff Faces corresponding to the mean vectors of Iris Setosa, Iris Versicolor and Iris Virginica.

the face associated with Iris Setosa looks quite different from the other two faces which implies that the Setosa category can be well separated from the remaining two categories in the four-dimensional feature space (This is also evident in the two-dimensional plots of this data in Fig. 5).

The main issue in dimensionality reduction is the choice of a criterion function. A commonly used criterion is the classification error of a feature subset. But, the classification error itself cannot be reliably estimated when the ratio of sample size to the number of features is small. In addition to the choice of a criterion function, we also need to determine the appropriate dimensionality of the reduced feature space. The answer to this question is embedded in the notion of the intrinsic dimensionality of data. Intrinsic dimensionality essentially determines whether the given d-dimensional patterns can be described adequately in a subspace of dimensionality less than d. For example, d-dimensional patterns along a reasonably smooth curve have an intrinsic dimensionality of one, irrespective of the value of d. Note that the intrinsic dimensionality is not the same as the linear dimensionality which is a global property of the data involving the number of significant eigenvalues of the covariance matrix of the data. While several algorithms are available to estimate the intrinsic dimensionality [81], they do not indicate how a subspace of the identified dimensionality can be easily identified.

We now briefly discuss some of the commonly used methods for feature extraction and feature selection.

4.1 Feature Extraction

Feature extraction methods determine an appropriate subspace of dimensionality m (either in a linear or a nonlinear way) in the original feature space of dimensionality d ($m \leq d$). Linear transforms, such as principal component analysis, factor analysis, linear discriminant analysis, and projection pursuit have been widely used in pattern recognition for feature extraction and dimensionality reduction. The best known linear feature extractor is the principal component analysis (PCA) or Karhunen-Loève expansion, that computes the m largest eigenvectors of the $d \times d$ covariance matrix of the n d-dimensional patterns. The linear transformation is defined as

$$Y = XH, \tag{7}$$

where X is the given $n \times d$ pattern matrix, Y is the derived $n \times m$ pattern matrix, and H is the $d \times m$ matrix of linear transformation whose columns are the eigenvectors. Since PCA uses the most expressive features (eigenvectors with the largest eigenvalues), it effectively approximates the data by a linear subspace using the mean squared error criterion. Other methods, like projection pursuit [53] and independent component analysis (ICA) [31], [11], [24], [96] are more appropriate for non-Gaussian distributions since they do not rely on the second-order property of the data. ICA has been successfully used for blind-source separation [78]; extracting linear feature combinations that define independent sources. This de-mixing is possible if at most one of the sources has a Gaussian distribution.

Whereas PCA is an unsupervised linear feature extraction method, discriminant analysis uses the category information associated with each pattern for (linearly) extracting the most discriminatory features. In discriminant analysis, inter-class separation is emphasized by replacing the total covariance matrix in PCA by a general separability measure like the Fisher criterion, which results in finding the eigenvectors of $S_w^{-1} S_b$ (the product of the inverse of the within-class scatter matrix, S_w, and the between-class scatter matrix, S_b) [58]. Another supervised criterion for non-Gaussian class-conditional densities is based on the Patrick-Fisher distance using Parzen density estimates [41].

There are several ways to define nonlinear feature extraction techniques. One such method which is directly related to PCA is called the Kernel PCA [73], [145]. The basic idea of kernel PCA is to first map input data into some new feature space F typically via a nonlinear function Φ (e.g., polynomial of degree p) and then perform a linear PCA in the mapped space. However, the F space often has a very high dimension. To avoid computing the mapping Φ explicitly, kernel PCA employs only Mercer kernels which can be decomposed into a dot product, $K(x, y) = \Phi(x) \cdot \Phi(y)$. As a result, the kernel space has a well-defined metric. Examples of Mercer kernels include p^{th}-order polynomial $(x-y)^p$ and Gaussian kernel $e^{-\frac{\|x-y\|^2}{c}}$.

Let X be the normalized $n \times d$ pattern matrix with zero mean, and $\Phi(X)$ be the pattern matrix in the F space. The linear PCA in the F space solves the eigenvectors of the correlation matrix $\Phi(X)\Phi(X)^T$, which is also called the kernel matrix $K(X,X)$. In kernel PCA, the first m eigenvectors of $K(X,X)$ are obtained to define a transformation matrix, E. (E has size $n \times m$, where m represents the desired number of features, $m \leq d$). New patterns $\boldsymbol{x}$ are mapped by $K(\boldsymbol{x}, X)E$, which are now represented relative to the training set and not by their measured feature values. Note that for a complete representation, up to m eigenvectors in E may be needed (depending on the kernel function) by kernel PCA, while in linear PCA a set of d eigenvectors represents the original feature space. How the kernel function should be chosen for a given application is still an open issue.

Multidimensional scaling (MDS) is another nonlinear feature extraction technique. It aims to represent a multidimensional dataset in 2 or 3 dimensions such that the distance matrix in the original d-dimensional feature space is preserved as faithfully as possible in the projected space. Various stress functions are used for measuring the performance of this mapping [20]; the most popular criterion is the stress function introduced by Sammon [141] and Niemann [114]. A problem with MDS is that it does not give an explicit mapping function, so it is not possible to place a new pattern in a map which has been computed for a given training set without repeating the mapping. Several techniques have been investigated to address this deficiency which range from linear interpolation to training a neural network [38]. It is also possible to redefine the MDS algorithm so that it directly produces a map that may be used for new test patterns [165].

A feed-forward neural network offers an integrated procedure for feature extraction and classification; the output of each hidden layer may be interpreted as a set of new, often nonlinear, features presented to the output layer for classification. In this sense, multi-layer networks serve as feature extractors [100]. For example, the networks used by Fukushima [62] and Le Cun [95] have the so called shared weight layers that are in fact filters for extracting features in two-dimensional images. During training, the filters are tuned to the data, so as to maximize the classification performance.

Neural networks can also be used directly for feature extraction in an unsupervised mode. Fig. 4(a) shows the architecture of a network which is able to find the PCA subspace [117]. Instead of sigmoids, the neurons have linear transfer functions. This

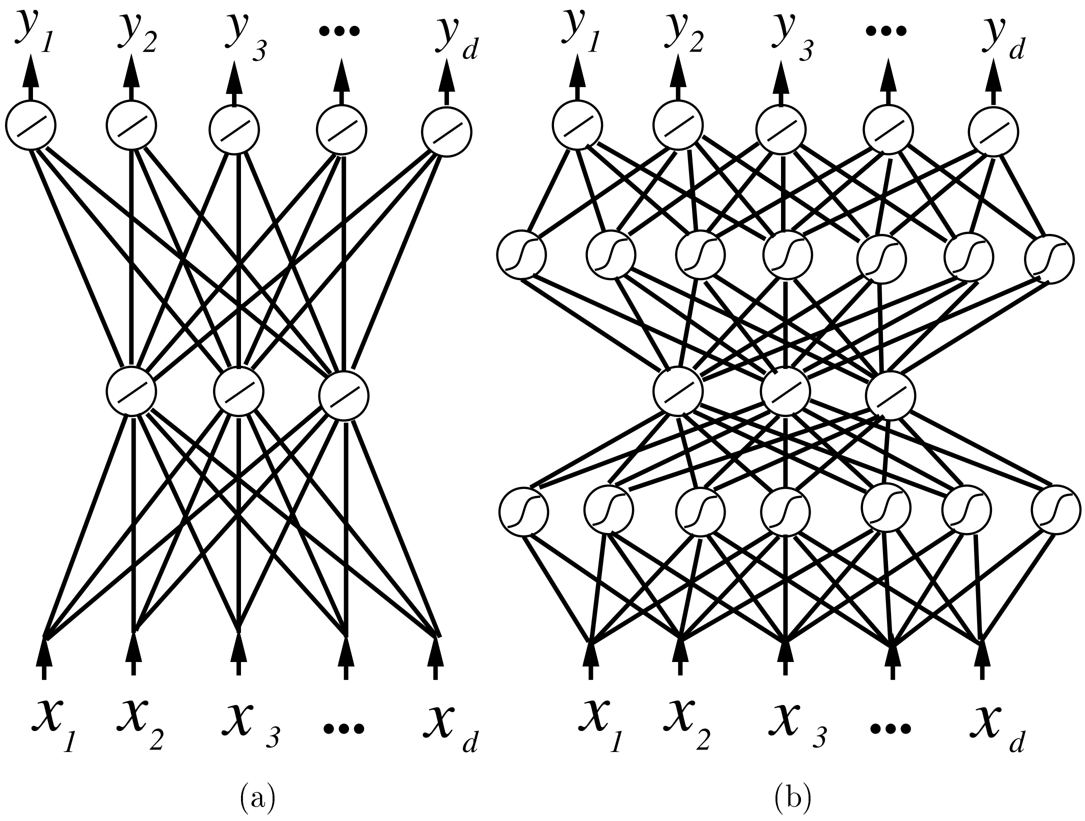


Figure 4: Auto-associative networks for finding a 3-dimensional subspace. (a) linear, (b) nonlinear (not all the connections are shown).

network has d inputs and d outputs, where d is the given number of features. The inputs are also used as targets, forcing the output layer to reconstruct the input space using only one hidden layer. The three nodes in the hidden layer capture the first three principal components [18]. If two nonlinear layers with sigmoidal hidden units are also included (see Fig. 4(b)), then a nonlinear subspace is found in the middle layer (also called the bottleneck layer). The nonlinearity is limited by the size of these additional layers. These so-called auto-associative, or nonlinear PCA networks offer a powerful tool to train and describe nonlinear subspaces [98]. Oja [118] shows how auto-associative networks can be used for ICA.

The Self-Organizing Map (SOM), or Kohonen Map [92], can also be used for nonlinear feature extraction. In SOM, neurons are arranged in an m-dimensional grid, where m is usually 1, 2, or 3. Each neuron is connected to all the d input units. The weights on the connections for each neuron form a d-dimensional weight vector. During training, patterns are presented to the network in a random order. At each presentation, the winner whose weight vector is the closest to the input vector is first identified. Then, all the neurons in the neighborhood (defined on the grid) of the winner are updated such that their weight vectors move towards the input vector. Consequently, after training is done, the weight vectors of neighboring neurons in the grid are likely to represent input patterns which are close in the original feature space. Thus, a "topology-preserving" map is formed. When the grid is plotted in the original space, the grid connections are more or less stressed according to the density of the training data. Thus, SOM offers an m-dimensional map with a spatial connectivity, which can be interpreted as feature extraction. SOM is different from learning vector quantization (LVQ) because no neighborhood is defined in LVQ.

Table 4 summarizes the feature extraction and projection methods discussed above. Note that the adjective nonlinear may be used both for the mapping (being a nonlinear

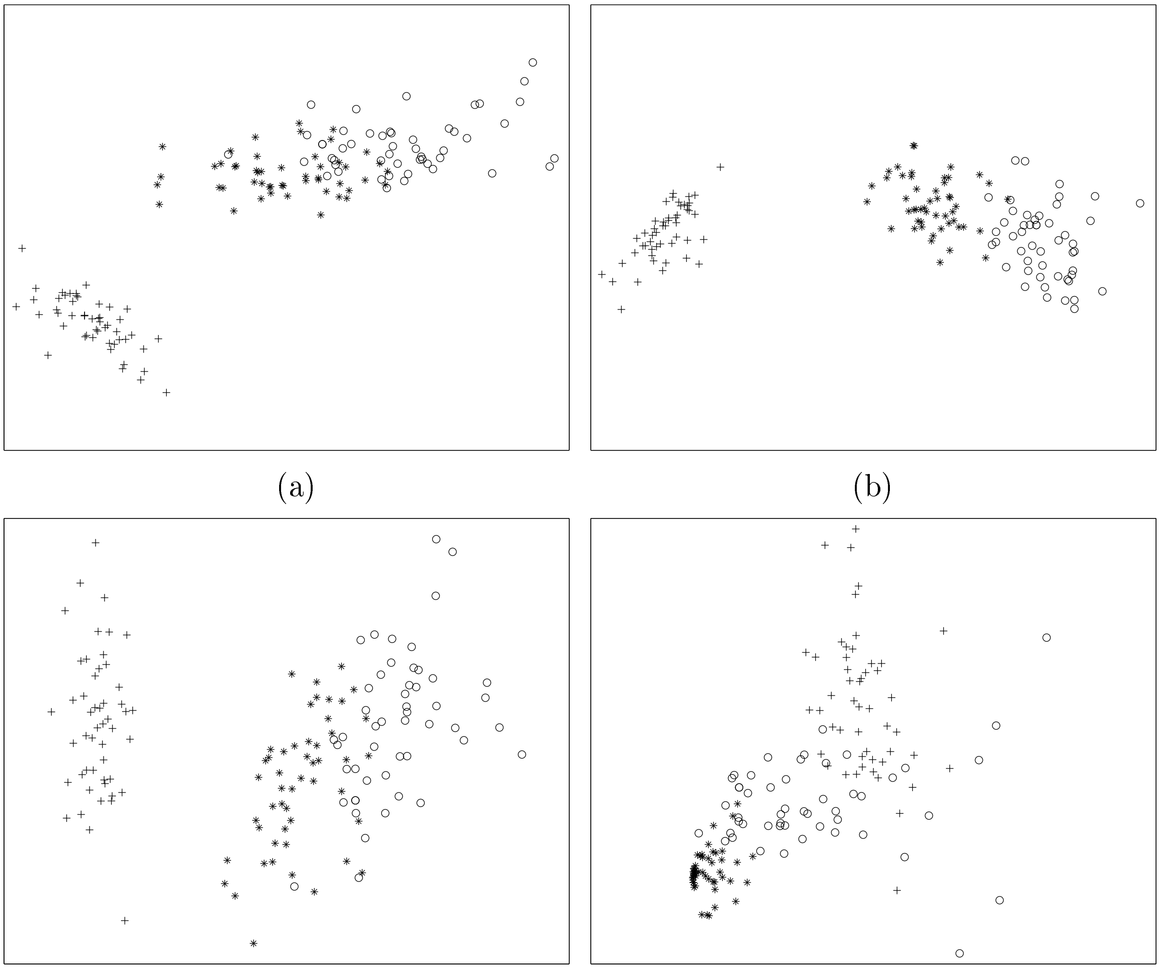
Table 4: FEATURE EXTRACTION AND PROJECTION METHODS

Method	Property	Comments
Principal Component Analysis (PCA)	Linear map; fast; eigenvector-based.	Traditional, eigenvector based method, also known as Karhunen-Loève expansion; good for Gaussian data.
Linear Discriminant Analysis	Supervised linear map; fast; eigenvector-based.	Better than PCA for classification; limited to $(c-1)$ components with non-zero eigenvalues.
Projection Pursuit	Linear map; iterative; non-Gaussian.	Mainly used for interactive exploratory data-analysis.
Independent Component Analysis (ICA)	Linear map, iterative, non-Gaussian.	Blind source separation, used for de-mixing non-Gaussian distributed sources (features).
Kernel PCA	Nonlinear map; eigenvector-based.	PCA-based method, using a kernel to replace inner products of pattern vectors.
PCA Network	Linear map; iterative.	Auto-associative neural network with linear transfer functions and just one hidden layer.
Nonlinear PCA	Linear map; non-Gaussian criterion; usually iterative	Neural network approach, possibly used for ICA.
Nonlinear auto-associative network	Nonlinear map; non-Gaussian criterion; iterative.	Bottleneck network with several hidden layers; the nonlinear map is optimized by a nonlinear reconstruction; input is used as target.
Multidimensional scaling (MDS), and Sammon's projection	Nonlinear map; iterative.	Iterative; often poor generalization; sample size limited; noise sensitive; mainly used for 2-dimensional visualization.
Self-Organizing Map (SOM)	Nonlinear; iterative.	Based on a grid of neurons in the feature space; suitable for extracting spaces of low dimensionality.

function of the original features) as well as for the criterion function (for non-Gaussian data). Fig. 5 shows an example of four different two-dimensional projections of the four-dimensional Iris dataset. Figs. 5(a) and (b) show two linear mappings, while Figs. 5(c) and (d) depict two nonlinear mappings. Only the Fisher mapping (Fig. 5(b)) makes use of the category information, this being the main reason why this mapping exhibits the best separation between the three categories.

4.2 Feature Selection

The problem of feature selection is defined as follows: given a set of d features, select a subset of size m that leads to the smallest classification error. There has been a resurgence



(a) (b) (c) (d)

Figure 5: Two-dimensional mappings of the Iris dataset (+: Iris Setosa; *: Iris Versicolor; o: Iris Virginica). (a) PCA, (b) Fisher Mapping, (c) Sammon Mapping, (d) Kernel PCA with second order polynomial kernel.

of interest in applying feature selection methods due to the large number of features encountered in the following situations: (i) multi-sensor fusion: features, computed from different sensor modalities, are concatenated to form a feature vector with a large number of components; (ii) integration of multiple data models: sensor data can be modeled using different approaches, where the model parameters serve as features, and the parameters from different models can be pooled to yield a high-dimensional feature vector.

Let Y be the given set of features, with cardinality d and let m represent the desired number of features in the selected subset X, $X \subseteq Y$. Let the feature selection criterion function for the set X be represented by $J(X)$. Let us assume that a higher value of J indicates a better feature subset; a natural choice for the criterion function is $J = (1 - P_e)$, where P_e denotes the classification error. The use of P_e in the criterion function makes feature selection procedures dependent on the specific classifier that is used and the sizes of the training and test sets. The most straightforward approach to the feature selection problem would require (i) examining all $\binom{d}{m}$ possible subsets of size m, and (ii) selecting the subset with the largest value of $J(\cdot)$. However, the number of possible subsets grows combinatorially, making this exhaustive search impractical for even moderate values of m and d. Cover and Van Campenhout [35] showed that no non-exhaustive sequential feature selection procedure can be guaranteed to produce the optimal subset. They further showed that any ordering of the classification errors of each of the 2^d feature subsets is possible. Therefore, in order to guarantee the optimality of, say, a 12-dimensional feature subset out of 24 available features, approximately 2.7 million possible subsets must be evaluated. The only "optimal" (in terms of a class of monotonic criterion functions) feature selection method which avoids the exhaustive search is based on the branch and bound algorithm. This procedure avoids an exhaustive search by using intermediate results for obtaining bounds on the final criterion value. The key to this algorithm is the monotonicity property of the criterion function $J(\cdot)$; given two features subsets X_1 and X_2, if $X_1 \subset X_2$, then $J(X_1) < J(X_2)$. In other words, the performance of a feature subset should improve whenever a feature is added to it. Most commonly used criterion functions do not satisfy this monotonicity property.

It has been argued that since feature selection is typically done in an off-line manner, the execution time of a particular algorithm is not as critical as the optimality of the feature subset it generates. While this is true for feature sets of moderate size, several recent applications, particularly those in data mining and document classification, involve thousands of features. In such cases, the computational requirement of a feature selection algorithm is extremely important. As the number of feature subset evaluations may easily become prohibitive for large feature sizes, a number of suboptimal selection techniques have been proposed which essentially tradeoff the optimality of the selected subset for computational efficiency.

Table 5 lists most of the well-known feature selection methods which have been proposed in the literature [85]. Only the first two methods in this table guarantee an optimal subset. All other strategies are suboptimal due to the fact that the best pair of features need not contain the best single feature [34]. More generally formulated: good larger feature sets do not necessarily include the good small sets. As a result, the simple method of selecting just the best individual features may fail dramatically. It might still be useful, however, as a first step in decreasing very large feature sets (e.g., hundreds of features). Further selection has to be done by more advanced methods that take feature dependencies into account. These operate either by evaluating growing feature sets (forward selection) or by evaluating shrinking feature sets (backward selection). A simple sequential method

Table 5: FEATURE SELECTION METHODS

Method	Property	Comments
Exhaustive Search	Evaluate all $\binom{d}{m}$ possible subsets.	Guaranteed to find the optimal subset; not feasible for even moderately large values of m and d.
Branch-and-Bound Search	Uses the well-known branch-and-bound search method; only a fraction of all possible feature subsets need to be enumerated to find the optimal subset.	Guaranteed to find the optimal subset provided the criterion function satisfies the monotonicity property; the worst-case complexity of this algorithm is exponential.
Best Individual Features	Evaluate all the m features individually; select the best m individual features.	Computationally simple; not likely to lead to an optimal subset.
Sequential Forward Selection (SFS)	Select the best single feature and then add one feature at a time which in combination with the selected features maximizes the criterion function.	Once a feature is retained, it cannot be discarded; computationally attractive since to select a subset of size 2, it examines only $(d-1)$ possible subsets.
Sequential Backward Selection (SBS)	Start with all the d features and successively delete one feature at a time.	Once a feature is deleted, it cannot be brought back into the optimal subset; requires more computation than sequential forward selection.
"Plus l-take away r" Selection	First enlarge the feature subset by l features using forward selection and then delete r features using backward selection.	Avoids the problem of feature subset "nesting" encountered in SFS and SBS methods; need to select values of l and $r(l > r)$.
Sequential Forward Floating Search (SFFS) and Sequential Backward Floating Search (SBFS)	A generalization of "plus-l take away-r" method; the values of l and r are determined automatically and updated dynamically.	Provides close to optimal solution at an affordable computational cost.

like SFS (SBS) adds (deletes) one feature at a time. More sophisticated techniques are the "Plus l - take away r" strategy and the Sequential Floating Search methods, SFFS and SBFS [126]. These methods backtrack as long as they find improvements compared to previous feature sets of the same size. In almost any large feature selection problem, these methods perform better than the straight sequential searches, SFS and SBS. SFFS and SBFS methods find 'nested' sets of features that remain hidden otherwise, but the number of feature set evaluations, however, may easily increase by a factor 2 to 10.

In addition to the search strategy, the user needs to select an appropriate evaluation criterion, $J(\cdot)$ and specify the value of m. Most feature selection methods use the classification error of a feature subset to evaluate its effectiveness. This could be done, for example, by a k-NN classifier using the leave-one-out method of error estimation. However, use of a different classifier and a different method for estimating the error rate could lead to a different feature subset being selected. Ferri et al. [50] and Jain and Zongker [85] have compared several of the feature selection algorithms in terms of classification error and run time. The general conclusion is that the sequential forward floating search (SFFS) method performs almost as well as the branch-and-bound algorithm and demands lower computational resources. Somol et al. [154] have proposed an adaptive version of the SFFS algorithm which has been shown to have superior performance.

The feature selection methods in Table 5 can be used with any of the well-known classifiers. But, if a multilayer feed forward network is used for pattern classification, then the node-pruning method simultaneously determines both the optimal feature subset and the optimal network classifier [26], [103]. First train a network, and then remove the least salient node (in input or hidden layers). The reduced network is trained again, followed by a removal of yet another least salient node. This procedure is repeated until the desired trade-off between classification error and size of the network is achieved. The pruning of an input node is equivalent to removing the corresponding feature.

How reliable are the feature selection results when the ratio of the available number of training samples to the number of features is small? Suppose the Mahalanobis distance [58] is used as the feature selection criterion. It depends on the inverse of the average class covariance matrix. The imprecision in its estimate in small sample size situations can result in an optimal feature subset which is quite different from the optimal subset that would be obtained when the covariance matrix is known. Jain and Zongker [85] illustrate this phenomenon for a two-class classification problem involving 20-dimensional Gaussian class-conditional densities (the same data was also used by Trunk [157] to demonstrate the curse of dimensionality phenomenon). As expected, the quality of the selected feature subset for small training sets is poor, but improves as the training set size increases. For example, with 20 patterns in the training set, the branch-and-bound algorithm selected a subset of 10 features which included only 5 features in common with the ideal subset of 10 features (when densities were known). With 2,500 patterns in the training set, the branch-and-bound procedure selected a 10-feature subset with only one wrong feature.

Fig. 6 shows an example of the feature selection procedure using the floating search technique on the PCA features in the digit dataset for two different training set sizes. The test set size is fixed at 1,000 patterns. In each of the selected feature spaces with dimensionalities ranging from 1 to 64, the Bayes plug-in classifier is designed assuming Gaussian densities with equal covariance matrices and evaluated on the test set. The feature selection criterion is the minimum pairwise Mahalanobis distance. In the small sample size case (total of 100 training patterns), the curse of dimensionality phenomenon can be clearly observed. In this case, the optimal number of features is about 20 which equals $n/5$ ($n = 100$), where n is the number of training patterns. The rule-of-thumb of having less than $n/10$ features is on the safe side in general.

5 Classifiers

Once a feature selection or classification procedure finds a proper representation, a classifier can be designed using a number of possible approaches. In practice, the choice of a classifier is a difficult problem and it is often based on which classifier(s) happen to

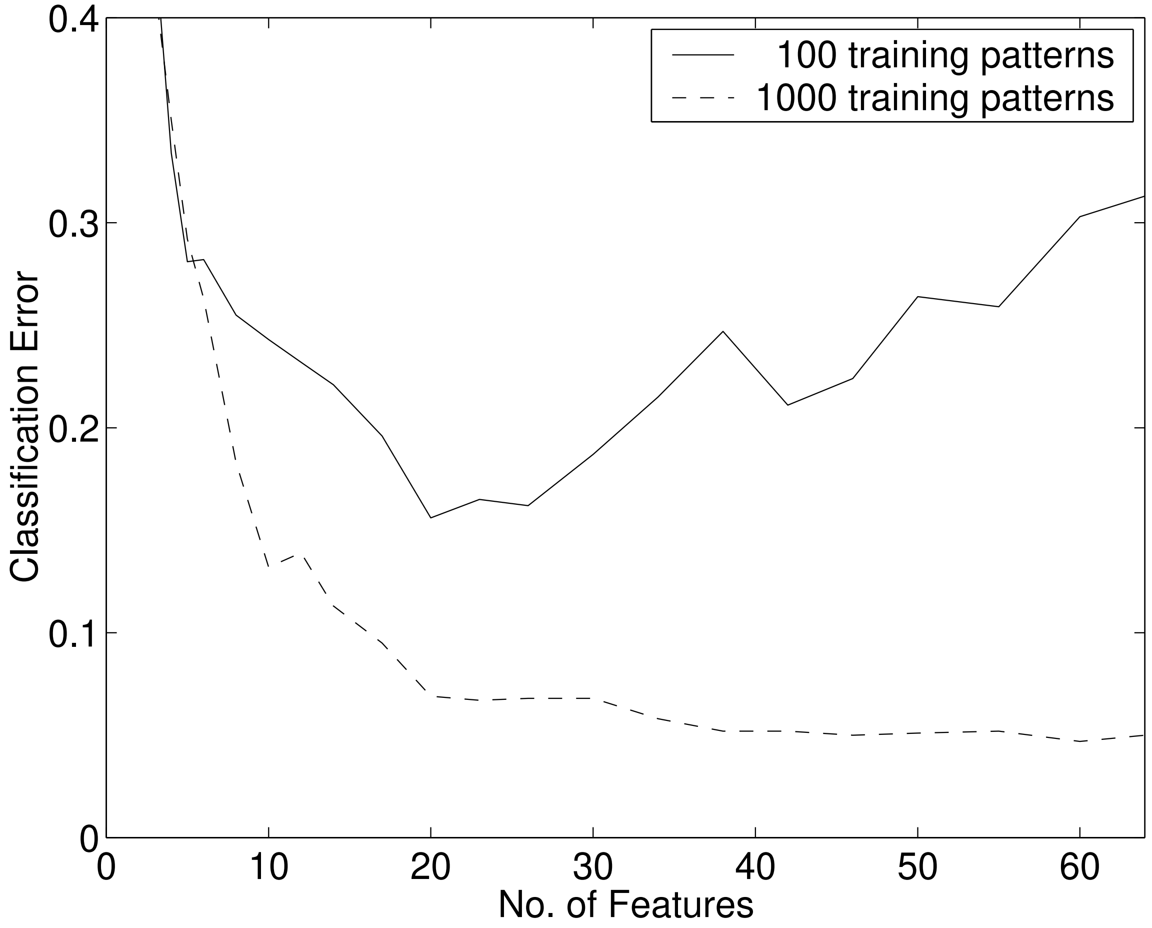


Figure 6: Classification error versus the number of features using the floating searching technique (see text).

be available, or best known, to the user.

We identify three different approaches to designing a classifier. The simplest and the most intuitive approach to classifier design is based on the concept of similarity: patterns that are similar should be assigned to the same class. So, once a good metric has been established to define similarity, patterns can be classified by template matching or the minimum distance classifier using a few prototypes per class. The choice of the metric and the prototypes is crucial to the success of this approach. In the nearest mean classifier, selecting prototypes is very simple and robust; each pattern class is represented by a single prototype which is the mean vector of all the training patterns in that class. More advanced techniques for computing prototypes are vector quantization [115], [171] and learning vector quantization [92], and the data reduction methods associated with the one-nearest neighbor decision rule (1-NN), such as editing and condensing [39]. The most straightforward 1-NN rule can be conveniently used as a benchmark for all the other classifiers since it appears to always provide a reasonable classification performance in most applications. Further, as the 1-NN classifier does not require any user-specified parameters (except perhaps the distance metric used to find the nearest neighbor, but Euclidean distance is commonly used), its classification results are implementation independent.

In many classification problems, the classifier is expected to have some desired invariant properties. An example is the shift invariance of characters in character recognition; a change in a character's location should not affect its classification. If the preprocessing or the representation scheme does not normalize the input pattern for this invariance, then the same character may be represented at multiple positions in the feature space. These positions define a one-dimensional subspace. As more invariants are considered, the dimensionality of this subspace correspondingly increases. Template matching or the nearest mean classifier can be viewed as finding the nearest subspace [116].

The second main concept used for designing pattern classifiers is based on the probabilistic approach. The optimal Bayes decision rule (with the 0/1 loss function) assigns a pattern to the class with the maximum posterior probability. This rule can be modified to take into account costs associated with different types of misclassifications. For known class conditional densities, the Bayes decision rule gives the optimum classifier, in the sense that, for given prior probabilities, loss function and class-conditional densities, no other decision rule will have a lower risk (i.e., expected value of the loss function, for example, probability of error). If the prior class probabilities are equal and a 0/1 loss function is adopted, the Bayes decision rule and the maximum likelihood decision rule exactly coincide. In practice, the empirical Bayes decision rule, or "plug-in" rule, is used: the estimates of the densities are used in place of the true densities. These density estimates are either parametric or nonparametric. Commonly used parametric models are multivariate Gaussian distributions [58] for continuous features, binomial distributions for binary features, and multinormal distributions for integer-valued (and categorical) features. A critical issue for Gaussian distributions is the assumption made about the covariance matrices. If the covariance matrices for different classes are assumed to be identical, then the Bayes plug-in rule, called Bayes-normal-linear, provides a linear decision boundary. On the other hand, if the covariance matrices are assumed to be different, the resulting Bayes plug-in rule, which we call Bayes-normal-quadratic, provides a quadratic decision boundary. In addition to the commonly used maximum likelihood estimator of the covariance matrix, various regularization techniques [54] are available to obtain a robust estimate in small sample size situations and the leave-one-out estimator is available for minimizing the bias [76].

A logistic classifier [4], which is based on the maximum likelihood approach, is well suited for mixed data types. For a two-class problem, the classifier maximizes:

$$\max_{\theta} \left\{ \prod_{\boldsymbol{x_i}^{(1)} \in \omega_1} q_1(\boldsymbol{x_i}^{(1)}; \theta) \prod_{\boldsymbol{x_i}^{(2)} \in \omega_2} q_1(\boldsymbol{x_i}^{(2)}; \theta) \right\}, \tag{8}$$

where $q_j(\boldsymbol{x}; \theta)$ is the posterior probability for pattern vector $\boldsymbol{x}$ given class ω_j, θ denotes the set of unknown parameters, and $\boldsymbol{x_i}^{(j)}$ denotes the i^{th} training sample from class ω_j, $j = 1, 2$. Given any discriminant function $D(\boldsymbol{x}; \theta)$, where θ is the parameter vector, the posterior probabilities can be derived as

$$q_1(\boldsymbol{x};\theta) = (1 + exp(-D(\boldsymbol{x};\theta)))^{-1}, q_2(\boldsymbol{x};\theta) = (1 + exp(D(\boldsymbol{x};\theta)))^{-1}, \tag{9}$$

which are called logistic functions. For linear discriminants, $D(\boldsymbol{x};\theta)$, Eq. (8) can be easily optimized. Equations (8) and (9) may also be used for estimating the class conditional posterior probabilities by optimizing $D(\boldsymbol{x};\theta)$ over the training set. The relationship between the discriminant function $D(\boldsymbol{x};\theta)$ and the posterior probabilities can be derived as follows. We know that the log-discriminant function for the Bayes decision rule, given the posterior probabilities $q_1(\boldsymbol{x};\theta)$ and $q_2(\boldsymbol{x};\theta)$, is $\log(q_1(\boldsymbol{x};\theta)/q_2(\boldsymbol{x};\theta))$. Assume that $D(\boldsymbol{x};\theta)$ can be optimized to approximate the Bayes decision boundary, i.e.,

$$D(\boldsymbol{x};\theta) = \log(q_1(\boldsymbol{x};\theta)/q_2(\boldsymbol{x};\theta)). \tag{10}$$

We also have

$$q_1(\boldsymbol{x};\theta) + q_2(\boldsymbol{x};\theta) = 1. \tag{11}$$

Solving Eqs. (10) and (11) for $q_1(\boldsymbol{x};\theta)$ and $q_2(\boldsymbol{x};\theta)$ results in Eq. (9).

The two well-known nonparametric decision rules, the k-nearest neighbor (k-NN) rule and the Parzen classifier (the class-conditional densities are replaced by their estimates using the Parzen window approach), while similar in nature, give different results in practice. They both have essentially one free parameter each, the number of neighbors k, or the smoothing parameter of the Parzen kernel, both of which can be optimized by a leave-one-out estimate of the error rate. Further, both these classifiers require the computation of the distances between a test pattern and all the patterns in the training set. The most convenient way to avoid these large numbers of computations is by a systematic reduction of the training set, e.g., by vector quantization techniques possibly combined with an optimized metric or kernel [60], [61]. Other possibilities like table-look-up and branch-and-bound methods [42] are less efficient for large dimensionalities.

The third category of classifiers is to construct decision boundaries (geometric approach in Fig. 2) directly by optimizing certain error criterion. While this approach depends on the chosen metric, sometimes classifiers of this type may approximate the Bayes classifier asymptotically. The driving force of the training procedure is, however, the minimization of a criterion such as the apparent classification error or the mean squared error (MSE) between the classifier output and some preset target value. A classical example of this type of classifier is Fisher's linear discriminant that minimizes the MSE between the classifier output and the desired labels. Another example is the single-layer perceptron, where the separating hyperplane is iteratively updated as a function of the distances of the misclassified patterns from the hyperplane. If the sigmoid function is used in combination with the MSE criterion, as in feed-forward neural nets (also called multi-layer perceptrons), the perceptron may show a behavior which is similar to other linear classifiers [133]. It is important to note that neural networks themselves can lead to many different classifiers depending on how they are trained. While the hidden layers in multi-layer perceptrons allow nonlinear decision boundaries, they also increase the danger of overtraining the classifier since the number of network parameters increases as more layers and more neurons per layer are added. Therefore, the regularization of neural networks may be necessary. Many regularization mechanisms are already built in, such as slow training in combination with early stopping. Other regularization methods include the addition of noise and weight decay [18, 28, 137], and also Bayesian learning [113].

One of the interesting characteristics of multi-layer perceptrons is that in addition to classifying an input pattern, they also provide a confidence in the classification, which is an approximation of the posterior probabilities. These confidence values may be used for rejecting a test pattern in case of doubt. The radial basis function (about a Gaussian kernel) is better suited than the sigmoid transfer function for handling outliers. A radial basis network, however, is usually trained differently than a multi-layer perceptron. Instead of a gradient search on the weights, hidden neurons are added until some preset performance is reached. The classification result is comparable to situations where each class conditional density is represented by a weighted sum of Gaussians (a so-called Gaussian mixture; see Section 8.2).

A special type of classifier is the decision tree [22, 30, 129], which is trained by an iterative selection of individual features that are most salient at each node of the tree. The criteria for feature selection and tree generation include the information content, the node purity, or Fisher's criterion. During classification, just those features are used that are needed for the test pattern under consideration, so feature selection is implicitly built-in. The most commonly used decision tree classifiers are binary in nature and use a single feature at each node, resulting in decision boundaries that are parallel to the feature axes [149]. Consequently, such decision trees are intrinsically suboptimal for most applications. However, the main advantage of the tree classifier, besides its speed, is the possibility to interpret the decision rule in terms of individual features. This makes decision trees attractive for interactive use by experts. Like neural networks, decision trees can be easily overtrained, which can be avoided by using a pruning stage [63], [106], [128]. Decision tree classification systems such as CART [22] and C4.5 [129] are available in the public domain[4] and therefore, often used as a benchmark.

One of the most interesting recent developments in classifier design is the introduction of the support vector classifier by Vapnik [162] which has also been studied by other authors [23], [144], [146]. It is primarily a two-class classifier. The optimization criterion here is the width of the margin between the classes, i.e. the empty area around the decision boundary defined by the distance to the nearest training patterns. These patterns, called support vectors, finally define the classification function. Their number is minimized by maximizing the margin.

The decision function for a two-class problem derived by the support vector classifier can be written as follows using a kernel function $K(\boldsymbol{x}_i, \boldsymbol{x})$ of a new pattern $\boldsymbol{x}$ (to be classified) and a training pattern $\boldsymbol{x}_i$.

$$D(\boldsymbol{x}) = \sum_{\forall \boldsymbol{x_i} \in S} \alpha_i \lambda_i K(\boldsymbol{x_i}, \boldsymbol{x}) + \alpha_0, \tag{12}$$

where S is the support vector set (a subset of the training set), and $\lambda_i = \pm 1$ the label of object $\boldsymbol{x}_i$. The parameters $\alpha_i \geq 0$ are optimized during training by

$$\min_\alpha (\alpha^T \Lambda K \Lambda \alpha + C \sum_j \varepsilon_j) \tag{13}$$

constrained by $\lambda_i D(\boldsymbol{x}_i) \geq 1 - \varepsilon_j$, $\forall \boldsymbol{x}_i$ in the training set. Λ is a diagonal matrix containing the labels λ_j and the matrix K stores the values of the kernel function $K(\boldsymbol{x_i}, \boldsymbol{x})$ for all pairs of training patterns. The set of slack variables ε_j allow for class overlap, controlled by the penalty weight $C > 0$. For $C = \infty$, no overlap is allowed. Equation 13 is the dual

[4] http://www.gmd.de/ml-archive/

form of maximizing the margin (plus the penalty term). During optimization, the values of all α_i become 0, except for the support vectors. So the support vectors are the only ones that are finally needed. The *ad hoc* character of the penalty term (error penalty) and the computational complexity of the training procedure (a quadratic minimization problem) are the drawbacks of this method. Various training algorithms have been proposed in the literature [23], including chunking [161], Osuna's decomposition method [119], and sequential minimal optimization [124]. An appropriate kernel function K (as in kernel PCA, Section 4.1) needs to be selected. In its most simple form, it is just a dot product between the input pattern $\boldsymbol{x}$ and a member of the support set: $K(\boldsymbol{x_i}, \boldsymbol{x}) = \boldsymbol{x_i} \cdot \boldsymbol{x}$, resulting in a linear classifier. Nonlinear kernels, such as $K(\boldsymbol{x_i}, \boldsymbol{x}) = (\boldsymbol{x_i} \cdot \boldsymbol{x} + 1)^p$, result in a p^{th}-order polynomial classifier. Gaussian radial basis functions can also be used. The important advantage of the support vector classifier is that it offers a possibility to train generalizable, nonlinear classifiers in high-dimensional spaces using a small training set. Moreover, for large training sets, it typically selects a small support set which is necessary for designing the classifier, thereby minimizing the computational requirements during testing.

The support vector classifier can also be understood in terms of the traditional template matching techniques. The support vectors replace the prototypes with the main difference being that they characterize the classes by a decision boundary. Moreover, this decision boundary is not just defined by the minimum distance function, but by a more general, possibly nonlinear, combination of these distances.

We summarize the most commonly used classifiers in Table 6. Many of them represent, in fact, an entire family of classifiers and allow the user to modify several associated parameters and criterion functions. All (or almost all) of these classifiers are admissible,

Table 6: Classification Methods

Method	Property	Comments
Template matching	Assign patterns to the most similar template.	The templates and the metric have to be supplied by the user; the procedure may include nonlinear normalizations; scale (metric) dependent.
Nearest Mean Classifier	Assign patterns to the nearest class mean.	Almost no training needed; fast testing; scale (metric) dependent.
Subspace Method	Assign patterns to the nearest class subspace.	Instead of normalizing on invariants, the subspace of the invariants is used; scale (metric) dependent.
1-Nearest Neighbor Rule	Assign patterns to the class of the nearest training pattern.	No training needed; robust performance; slow testing; scale (metric) dependent.
k-Nearest Neighbor Rule	Assign patterns to the majority class among k nearest neighbor using a performance optimized value for k.	Asymptotically optimal; scale (metric) dependent; slow testing.
Bayes plug-in	Assign pattern to the class which has the maximum estimated posterior probability.	Yields simple classifiers (linear or quadratic) for Gaussian distributions; sensitive to density estimation errors.
Logistic Classifier	Maximum likelihood rule for logistic (sigmoidal) posterior probabilities.	Linear classifier; iterative procedure; optimal for a family of different distributions (Gaussian); suitable for mixed data types.
Parzen Classifier	Bayes plug-in rule for Parzen density estimates with performance optimized kernel.	Asymptotically optimal; scale (metric) dependent; slow testing.
Fisher Linear Discriminant	Linear classifier using MSE optimization.	Simple and fast; similar to Bayes plug-in for Gaussian distributions with identical covariance matrices.
Binary Decision Tree	Finds a set of thresholds for a pattern-dependent sequence of features.	Iterative training procedure; overtraining sensitive; needs pruning; fast testing.
Perceptron	Iterative optimization of a linear classifier.	Sensitive to training parameters; may produce confidence values.
Multi-layer Perceptron (Feed-Forward Neural Network)	Iterative MSE optimization of two or more layers of perceptrons (neurons) using sigmoid transfer functions.	Sensitive to training parameters; slow training; nonlinear classification function; may produce confidence values; overtraining sensitive; needs regularization.
Radial Basis Network	Iterative MSE optimization of a feed-forward neural network with at least one layer of neurons using Gaussian-like transfer functions.	Sensitive to training parameters; nonlinear classification function; may produce confidence values; overtraining sensitive; needs regularization; may be robust to outliers.
Support Vector Classifier	Maximizes the margin between the classes by selecting a minimum number of support vectors.	Scale (metric) dependent; iterative; slow training; nonlinear; overtraining insensitive; good generalization performance.

in the sense that there exist some classification problems for which they are the best choice. An extensive comparison of a large set of classifiers over many different problems is the StatLog project [109] which showed a large variability over their relative performances, proving that there is no such thing as an overall optimal classification rule.

The differences between the decision boundaries obtained by different classifiers are illustrated in Fig. 7 using dataset 1 (2-dimensional, two-class problem with Gaussian densities). Note the two small isolated areas for R_1 in Fig. 7(c) for the 1-NN rule. The neural network classifier in Fig. 7(d) even shows a 'ghost' region that seemingly has nothing to do with the data. Such regions are less probable for a small number of hidden layers at the cost of poorer class separation.

A larger hidden layer may result in overtraining. This is illustrated in Fig. 8 for a network with 10 neurons in the hidden layer. During training, the test set error and the training set error are initially almost equal, but after a certain point (3 epochs [5]) the test set error starts to increase while the training error keeps on decreasing. The final classifier after 50 epochs has clearly adapted to the noise in the dataset: it tries to separate isolated patterns in a way that does not contribute to its generalization ability.

6 Classifier Combination

There are several reasons for combining multiple classifiers to solve a given classification problem. Some of them are listed below.

(i) A designer may have access to a number of different classifiers, each developed in a different context and for an entirely different representation/description of the same

[5] One epoch means going through the entire training data once.

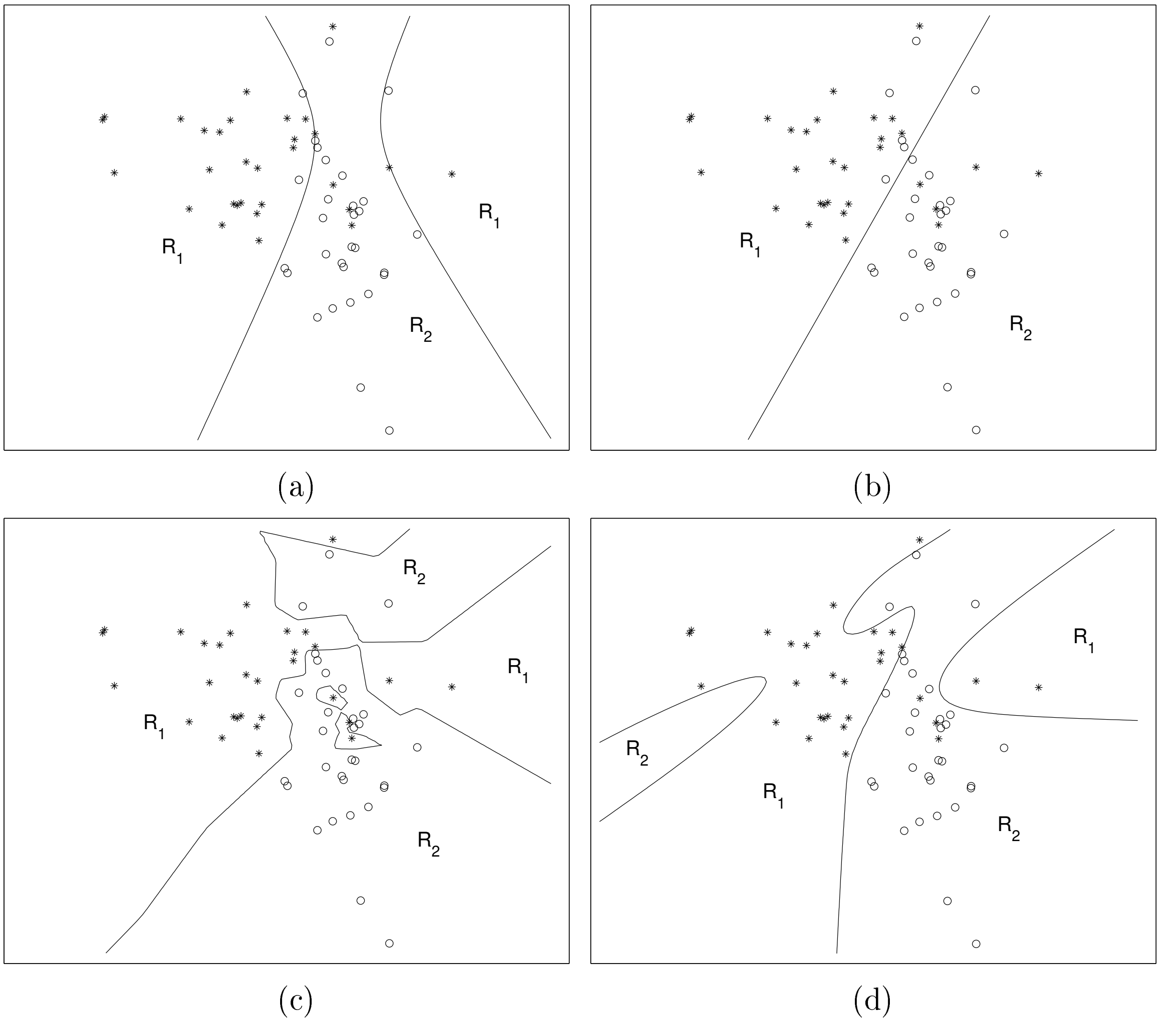


Figure 7: Decision boundaries for two bivariate Gaussian distributed classes, using 30 patterns per class. The following classifiers are used: (a) Bayes-normal-quadratic, (b) Bayes-normal-linear, (c) 1-NN, (d) ANN-5 (a feed-forward neural network with one hidden layer containing 5 neurons. The regions R_1 and R_2 for classes ω_1 and ω_2, respectively, are found by classifying all the points in the 2-dimensional feature space.

problem. An example is the identification of persons by their voice, face, as well as handwriting.

(ii) Sometimes more than a single training set is available, each collected at a different time or in a different environment. These training sets may even use different features.

(iii) Different classifiers trained on the same data may not only differ in their global performances, but they also may show strong local differences. Each classifier may have its own region in the feature space where it performs the best.

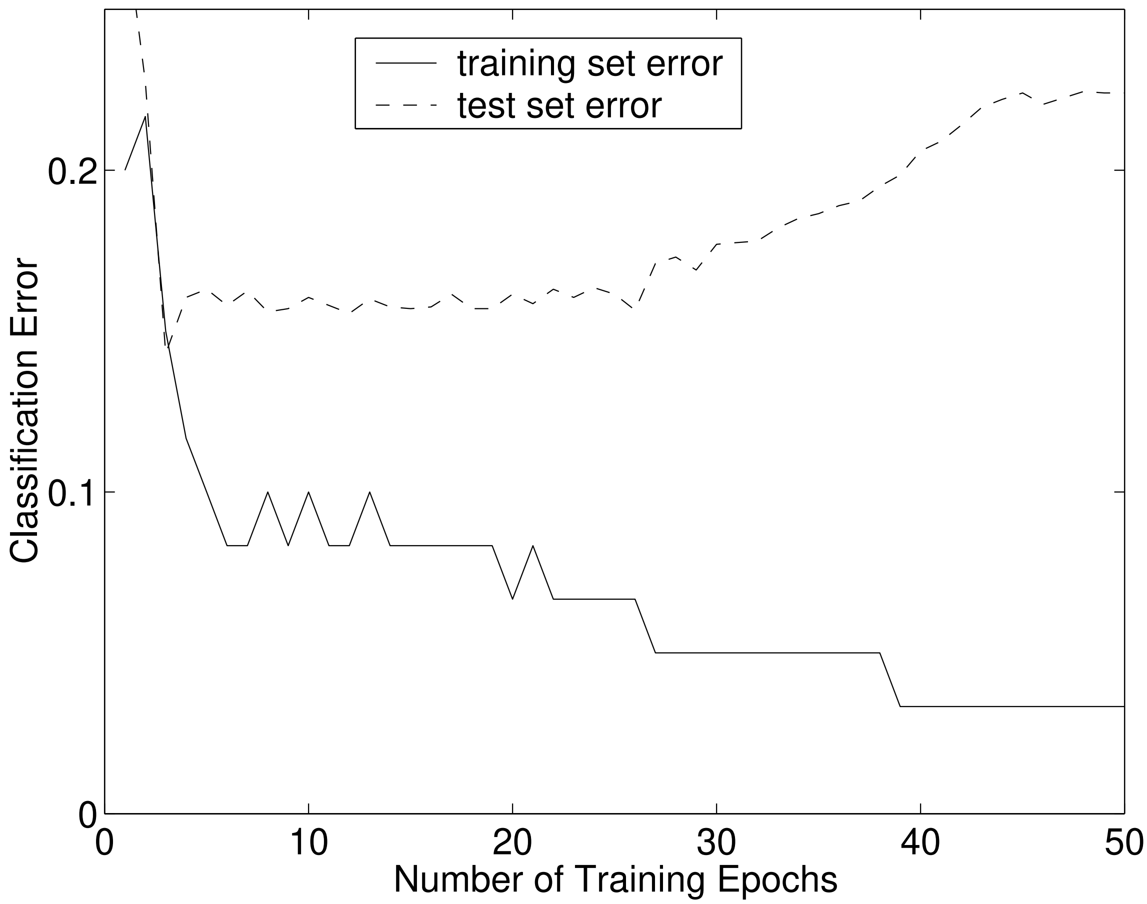


Figure 8: Classification error of a neural network classifier using 10 hidden units trained by the Levenberg-Marquardt rule for 50 epochs from two classes with 30 patterns each (Dataset 1). Test set error is based on an independent set of 1000 patterns.

(iv) Some classifiers such as neural networks show different results with different initializations due to the randomness inherent in the training procedure. Instead of selecting the best network and discarding the others, one can combine various networks, thereby taking advantage of all the attempts to learn from the data.

In summary, we may have different feature sets, different training sets, different classification methods or different training sessions, all resulting in a set of classifiers whose outputs may be combined, with the hope of improving the overall classification accuracy. If this set of classifiers is fixed, the problem focuses on the combination function. It is also possible to use a fixed combiner and optimize the set of input classifiers, see Section

6.1.

A large number of combination schemes have been proposed in the literature [172]. A typical combination scheme consists of a set of individual classifiers and a combiner which combines the results of the individual classifiers to make the final decision. When the individual classifiers should be invoked or how they should interact with each other is determined by the architecture of the combination scheme. Thus, various combination schemes may differ from each other in their architectures, the characteristics of the combiner, and selection of the individual classifiers.

Various schemes for combining multiple classifiers can be grouped into three main categories according to their architecture: (i) parallel, (ii) cascading (or serial combination), and (iii) hierarchical (tree-like). In the parallel architecture, all the individual classifiers are invoked independently, and their results are then combined by a combiner. Most combination schemes in the literature belong to this category. In the gated parallel variant, the outputs of individual classifiers are selected or weighted by a gating device before they are combined. In the cascading architecture, individual classifiers are invoked in a linear sequence. The number of possible classes for a given pattern is gradually reduced as more classifiers in the sequence have been invoked. For the sake of efficiency, inaccurate but cheap classifiers (low computational and measurement demands) are considered first, followed by more accurate and expensive classifiers. In the hierarchical architecture, individual classifiers are combined into a structure, which is similar to that of a decision tree classifier. The tree nodes, however, may now be associated with complex classifiers demanding a large number of features. The advantage of this architecture is the high efficiency and flexibility in exploiting the discriminant power of different types of features. Using these three basic architectures, we can build even more complicated classifier combination systems.

6.1 Selection and Training of Individual Classifiers

A classifier combination is especially useful if the individual classifiers are largely independent. If this is not already guaranteed by the use of different training sets, various resampling techniques like rotation and bootstrapping may be used to artificially create such differences. Examples are stacking [168], bagging [21], and boosting (or ARCing) [142]. In stacking, the outputs of the individual classifiers are used to train the "stacked" classifier. The final decision is made based on the outputs of the stacked classifier in conjunction with the outputs of individual classifiers.

In bagging, different datasets are created by bootstrapped versions of the original dataset and combined using a fixed rule like averaging. Boosting [52] is another resampling technique for generating a sequence of training data sets. The distribution of a particular training set in the sequence is overrepresented by patterns which were misclassified by the earlier classifiers in the sequence. In boosting, the individual classifiers are trained hierarchically to learn to discriminate more complex regions in the feature space. The original algorithm was proposed by Schapire [142], who showed that, in principle, it is possible for a combination of *weak* classifiers (whose performances are only slightly better than random guessing) to achieve an error rate which is arbitrarily small on the training data.

Sometimes cluster analysis may be used to separate the individual classes in the training set into subclasses. Consequently, simpler classifiers (e.g., linear) may be used and combined later to generate, for instance, a piece-wise linear result [120].

Instead of building different classifiers on different sets of training patterns, different feature sets may be used. This even more explicitly forces the individual classifiers to contain independent information. An example is the random subspace method [75].

6.2 Combiner

After individual classifiers have been selected, they need to be combined together by a module, called the combiner. Various combiners can be distinguished from each other in their trainability, adaptivity, and requirement on the output of individual classifiers. Combiners, such as voting, averaging (or sum), and Borda count [74] are static, with no training required, while others are trainable. The trainable combiners may lead to a better improvement than static combiners at the cost of additional training as well as the requirement of additional training data.

Some combination schemes are adaptive in the sense that the combiner evaluates (or weighs) the decisions of individual classifiers depending on the input pattern. In contrast, non-adaptive combiners treat all the input patterns the same. Adaptive combination schemes can further exploit the detailed error characteristics and expertise of individual classifiers. Examples of adaptive combiners include adaptive weighting [156], associative switch, mixture of local experts (MLE) [79], and hierarchical MLE [87].

Different combiners expect different types of output from individual classifiers. Xu et al. [172] grouped these expectations into three levels: (i) measurement (or confidence), (ii) rank, and (iii) abstract. At the confidence level, a classifier outputs a numerical value for each class indicating the belief or probability that the given input pattern belongs to that class. At the rank level, a classifier assigns a rank to each class with the highest rank being the first choice. Rank value can not be used in isolation because the highest rank does not necessarily mean a high confidence in the classification. At the abstract level, a classifier only outputs a unique class label or several class labels (in which case, the classes are equally good). The confidence level conveys the richest information, while the abstract level contains the least amount of information about the decision being made.

Table 7 lists a number of representative combination schemes and their characteristics. This is by no means an exhaustive list.

Table 7: Classifier Combination schemes

Scheme	Architecture	Trainable	Adaptive	Info-level	Comments
Voting	Parallel	No	No	Abstract	Assumes independent classifiers
Sum, mean, median	Parallel	No	No	Confidence	Robust; assumes independent confidence estimators
Product, min, max	Parallel	No	No	Confidence	Assumes independent features
Generalized ensemble	Parallel	Yes	No	Confidence	Considers error correlation
Adaptive weighting	Parallel	Yes	Yes	Confidence	Explores local expertise
Stacking	Parallel	Yes	No	Confidence	Good utilization of training data
Borda count	Parallel	Yes	No	Rank	Converts ranks into confidences
Logistic regression	Parallel	Yes	No	Rank confidence	Converts ranks into confidences
Class set reduction	Parallel cascading	Yes / No	No	Rank confidence	Efficient
Dempster-Shafer	Parallel	Yes	No	Rank confidence	Fuses non-probabilistic confidences
Fuzzy integrals	Parallel	Yes	No	Confidence	Fuses non-probabilistic confidences
Mixture of local experts (MLE)	Gated parallel	Yes	Yes	Confidence	Explores local expertise; joint optimization
Hierarchical MLE	Gated parallel hierarchical	Yes	Yes	Confidence	Same as MLE; hierarchical
Associative switch	Parallel	Yes	Yes	Abstract	Same as MLE, but no joint optimization
Bagging	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Boosting	Parallel hierarchical	Yes	No	Abstract	Improves margins; unlikely to overtrain; sensitive to mislabels; needs many comparable classifiers
Random subspace	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Neural tree	Hierarchical	Yes	No	Confidence	Handles large numbers of classes

6.3 Theoretical Analysis of Combination Schemes

A large number of experimental studies have shown that classifier combination can improve the recognition accuracy. However, there exist only a few theoretical explanations for these experimental results. Moreover, most explanations apply to only the simplest combination schemes under rather restrictive assumptions. One of the most rigorous theories on classifier combination is presented by Kleinberg [91].

A popular analysis of combination schemes is based on the well-known bias-variance dilemma [64, 93]. Regression or classification error can be decomposed into a bias term and a variance term. Unstable classifiers or classifiers with a high complexity (or capacity), such as decision trees, nearest neighbor classifiers, and large-size neural networks, can have universally low bias, but a large variance. On the other hand, stable classifiers or classifiers with a low capacity can have a low variance but a large bias.

Tumer and Ghosh [158] provided a quantitative analysis of the improvements in classification accuracy by combining multiple neural networks. They showed that combining networks using a linear combiner or order statistics combiner reduces the variance of the actual decision boundaries around the optimum boundary. In the absence of network bias, the reduction in the added error (to Bayes error) is directly proportional to the reduction in the variance. A linear combination of N unbiased neural networks with independent and identically distributed (i.i.d) error distributions can reduce the variance by a factor of N. At a first glance, this result sounds remarkable for as N approaches infinity, the variance is reduced to zero. Unfortunately, this is not realistic because the i.i.d assumption breaks down for large N. Similarly, Perrone and Cooper [123] showed that under the zero-mean and independence assumption on the misfit (difference between the desired output and the actual output), averaging the outputs of N neural networks can reduce the mean square error (MSE) by a factor of N compared to the averaged MSE of the N neural networks. For a large N, the MSE of the ensemble can, in principle, be made arbitrarily small. Unfortunately, as mentioned above, the independence assumption breaks down as N increases. Perrone and Cooper [123] also proposed a generalized ensemble, an optimal linear combiner in the least square error sense. In the generalized ensemble, weights are derived from the error correlation matrix of the N neural networks. It was shown that the MSE of the generalized ensemble is smaller than the MSE of the best neural network in the ensemble. This result is based on the assumptions that the rows and columns of the error correlation matrix are linearly independent and the error correlation matrix can be reliably estimated. Again, these assumptions break down as N increases.

Kittler et al. [90] developed a common theoretical framework for a class of combination schemes where individual classifiers use distinct features to estimate the posterior probabilities given the input pattern. They introduced a sensitivity analysis to explain why the sum (or average) rule outperforms the other rules for the same class. They showed that the sum rule is less sensitive than others (such as the "product" rule) to the error of individual classifiers in estimating posterior probabilities. The sum rule is most appropriate for combining different estimates of the same posterior probabilities, e.g. resulting from different classifier initializations (case (iv) in the introduction of this chapter). The product rule is most appropriate for combining preferably error-free independent probabilities, e.g. resulting from well estimated densities of different, independent feature sets (case (ii) in the introduction of this chapter).

Schapire et al. [143] proposed a different explanation for the effectiveness of voting (weighted average, in fact) methods. The explanation is based on the notion of "margin" which is the difference between the combined score of the correct class and the highest combined score among all the incorrect classes. They established that the generalization error is bounded by the tail probability of the margin distribution on training data plus a term which is a function of the complexity of a single classifier rather than the combined classifier. They demonstrated that the boosting algorithm can effectively improve the margin distribution. This finding is similar to the property of the support vector classifier, which shows the importance of training patterns near the margin, where the margin is defined as the area of overlap between the class conditional densities.

6.4 An Example

We will illustrate the characteristics of a number of different classifiers and combination rules on a digit classification problem (Dataset 3, see Section 2). The classifiers used in the experiment were designed using Matlab and were not optimized for the data set. All the six different feature sets for the digit dataset discussed in Section 2 will be used, enabling us to illustrate the performance of various classifier combining rules over different classifiers as well as over different feature sets. Confidence values in the outputs of all the classifiers are computed, either directly based on the posterior probabilities or on the logistic output function as discussed in Section 5. These outputs are also used to obtain multi-class versions for intrinsically two-class discriminants such as the Fisher Linear Discriminant and the Support Vector Classifier (SVC). For these two classifiers, a total of 10 discriminants are computed between each of the 10 classes and the combined set of the remaining classes. A test pattern is classified by selecting the class for which the discriminant has the highest confidence.

The following 12 classifiers are used (see also Table 8): the Bayes-plug-in rule assuming normal distributions with different (Bayes-normal-quadratic) or equal covariance matrices (Bayes-normal-linear), the Nearest Mean (NM) rule, 1-NN, k-NN, Parzen, Fisher, a binary decision tree using the maximum purity criterion [21] and early pruning, two feed-forward neural networks (based on the Matlab Neural Network Toolbox) with a hidden layer consisting of 20 (ANN-20) and 50 (ANN-50) neurons and the linear (SVC-linear) and quadratic (SVC-quadratic) Support Vector classifiers. The number of neighbors in the k-NN rule and the smoothing parameter in the Parzen classifier are both optimized over the classification result using the leave-one-out error estimate on the training set. For combining classifiers, the median, product and voting rules are used, as well as two trained classifiers (NM and 1-NN). The training set used for the individual classifiers is also used in classifier combination.

The 12 classifiers listed in Table 8 were trained on the same 500 (10×50) training patterns from each of the 6 feature sets and tested on the same 1,000 (10×100) test patterns. The resulting classification errors (in %) are reported; for each feature set, the best result over the classifiers is printed in bold. Next, the 12 individual classifiers for a single feature set were combined using the five combining rules (median, product, voting, nearest mean and 1-NN). For example, the voting rule (row) over the classifiers using feature set No. 3 (column) yields an error of 3.2%. It is underlined to indicate that this combination result is better than the performance of individual classifiers for this feature set. Finally, the outputs of each classifier and each classifier combination scheme over all the six feature sets are combined using the 5 combination rules (last 5 columns). For example, the voting rule (column) over the six decision tree classifiers (row) yields an error of 21.8%. Again, it is underlined to indicate that this combination result is better than each of the six individual results of the decision tree. The 5×5 block in the bottom

Table 8: Error Rates (in %) of Different Classifiers and Classifier Combination Schemes

	Feature set (see Section 2)						Combination Rule				
Classifier / Combining rule	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	Med.	Prod.	Voting	NM	1-NN
Bayes-normal-quadratic	25.7	5.8	12.8	6.2	21.2	31.0	2.8	6.3	6.8	6.7	5.0
Bayes-normal-linear	21.3	**3.4**	5.7	9.9	**18.0**	29.1	3.7	3.1	5.1	3.9	4.2
Nearest Mean	22.4	18.1	9.9	9.6	27.8	54.0	6.2	4.6	7.5	10.3	4.6
1-NN	19.2	9.0	4.4	**3.7**	19.7	57.0	2.6	1.7	4.0	11.3	3.0
k-NN	18.9	9.2	4.4	**3.7**	19.3	51.0	5.4	4.2	5.1	3.6	2.6
Parzen	**17.1**	7.9	**3.7**	**3.7**	18.5	52.1	2.9	2.7	5.1	3.1	3.1
Fisher	24.8	4.7	8.2	15.3	21.0	**28.2**	3.2	5.2	5.7	3.5	3.6
Dec. Tree	45.4	40.3	40.0	54.9	59.8	32.9	13.4	11.0	21.8	10.2	10.8
ANN-20	90.0	4.6	14.6	85.2	90.0	32.8	17.7	90.0	32.7	2.6	2.1
ANN-50	24.5	13.0	82.3	81.0	26.5	71.7	24.4	80.7	16.3	5.5	3.3
SVC-linear	24.6	6.6	6.1	7.7	29.4	84.8	10.8	10.1	4.7	6.0	5.8
SVC-quadratic	21.2	5.1	4.0	6.0	19.3	81.1	3.6	3.8	3.8	4.0	4.0
Median	19.0	4.3	3.6	4.5	17.4	28.7	2.3	2.5	5.0	1.9	5.0
Product	29.4	13.1	4.4	8.2	40.1	41.2	23.4	8.6	56.8	85.1	68.5
Voting	17.5	3.5	3.2	3.7	16.9	31.8	2.3	2.0	4.8	2.1	2.0
Nearest Mean	19.8	3.7	4.6	7.3	18.1	26.6	2.0	1.9	5.1	1.5	1.8
1-NN	18.6	3.8	4.1	7.2	17.0	32.8	1.8	1.8	4.1	1.9	1.8

right part of Table 8 presents the combination results, over the six features sets, for the classifier combination schemes for each of the separate feature sets.

Some of the classifiers, for example, the decision tree, do not perform well on this data. Also, the neural network classifiers provide rather poor optimal solutions, probably due to non-converging training sessions. Some of the simple classifiers such as the 1-NN, Bayes plug-in, and Parzen give good results; the performances of different classifiers vary substantially over different feature sets. Due to the relatively small training set for some of the large feature sets, the Bayes-normal-quadratic classifier is outperformed by the linear one, but the SVC-quadratic generally performs better than the SVC-linear. This shows that the SVC classifier can find nonlinear solutions without increasing the overtraining risk.

Considering the classifier combination results, it appears that the trained classifier combination rules are not always better than the use of fixed rules. Still, the best overall result (1.5% error) is obtained by a trained combination rule, the nearest mean method. The combination of different classifiers for the same feature set (columns in the table) only slightly improves the best individual classification results. The best combination rule for this dataset is voting. The product rule behaves poorly, as can be expected, because different classifiers on the same feature set do not provide independent confidence values. The combination of results obtained by the same classifier over different feature sets (rows in the table) frequently outperforms the best individual classifier result. Sometimes, the improvements are substantial as is the case for the decision tree. Here, the product rule does much better, but occasionally it performs surprisingly bad, similar to the combination of neural network classifiers. This combination rule (like the minimum and maximum rules, not used in this experiment) is sensitive to poorly trained individual classifiers. Finally, it is worthwhile to observe that in combining the neural network results, the trained combination rules do very well (classification errors between 2.1% and 5.6%) in comparison with the fixed rules (classification errors between 16.3% to 90%).

7 Error Estimation

The classification error or simply the error rate, P_e, is the ultimate measure of the performance of a classifier. Competing classifiers can also be evaluated based on their error probabilities. Other performance measures include the cost of measuring features and the computational requirements of the decision rule. While it is easy to define the probability of error in terms of the class-conditional densities, it is very difficult to obtain a closed-form expression for P_e. Even in the relatively simple case of multivariate Gaussian densities with unequal covariance matrices, it is not possible to write a simple analytical expression for the error rate. If an analytical expression for the error rate was available, it could be used, for a given decision rule, to study the behavior of P_e as a function of the number of features, true parameter values of the densities, number of training samples, and prior class probabilities. For consistent training rules the value of P_e approaches the Bayes error for increasing sample sizes. For some families of distributions tight bounds for the Bayes error may be obtained [7]. For finite sample sizes and unknown distributions, however, such bounds are impossible [6, 41].

In practice, the error rate of a recognition system must be estimated from all the available samples which are split into training and test sets [70]. The classifier is first designed using training samples, and then it is evaluated based on its classification performance on the test samples. The percentage of misclassified test samples is taken as an estimate of the error rate. In order for this error estimate to be reliable in predicting future classification performance, not only should the training set and the test set be sufficiently large, but the training samples and the test samples must be independent. This requirement of independent training and test samples is still often overlooked in practice.

An important point to keep in mind is that the error estimate of a classifier, being a function of the specific training and test sets used, is a random variable. Given a classifier, suppose τ is the number of test samples (out of a total of n) that are misclassified. It can be shown that the probability density function of τ has a binomial distribution. The maximum-likelihood estimate, $\hat{P}_e$, of P_e is given by $\hat{P}_e = \tau/n$, with $E(\hat{P}_e) = P_e$ and $Var(\hat{P}_e) = P_e(1 - P_e)/n$. Thus $\hat{P}_e$ is an unbiased and consistent estimator. Because $\hat{P}_e$ is a random variable, a confidence interval is associated with it. Suppose $n = 250$ and

$\tau = 50$ then $\hat{P}_e = 0.2$ and a 95% confidence interval of $\hat{P}_e$ is $(0.15, 0.25)$. The confidence interval, which shrinks as the number n of test samples increases, plays an important role in comparing two competing classifiers, C_1 and C_2. Suppose a total of 100 test samples are available and C_1 and C_2 misclassify 10 and 13, respectively, of these samples. Is classifier C_1 better than C_2? The 95% confidence intervals for the true error probabilities of these classifiers are (0.04, 0.16) and (0.06, 0.20), respectively. Since these confidence intervals overlap, we cannot say that the performance of C_1 will always be superior to that of C_2. This analysis is somewhat pessimistic due to positively corrected error estimates based on the same test set [137].

How should the available samples be split to form training and test sets? If the training set is small, then the resulting classifier will not be very robust and will have a low generalization ability. On the other hand, if the test set is small, then the confidence in the estimated error rate will be low. Various methods that are commonly used to estimate the error rate are summarized in Table 9. These methods differ in how they utilize the available samples as training and test sets. If the number of available samples is extremely large (say, 1 million), then all these methods are likely to lead to the same estimate of the error rate. For example, while it is well known that the resubstitution method provides an optimistically biased estimate of the error rate, the bias becomes smaller and smaller as the ratio of the number of training samples per class to the dimensionality of the feature vector gets larger and larger. There are no good guidelines available on how to divide the available samples into training and test sets; Fukunaga [58] provides arguments in favor of using more samples for testing the classifier than for designing the classifier. No matter how the data is split into training and test sets, it should be clear that different random splits (with the specified size of training and test sets) will result in different error estimates.

Table 9: Error Estimation Methods

Method	Property	Comments
Resubstitution Method	All the available data is used for training as well as testing; training and test sets are the same.	Optimistically biased estimate, especially when the ratio of sample size to dimensionality is small.
Holdout Method	Half the data is used for training and the remaining data is used for testing; training and test sets are independent.	Pessimistically biased estimate; different partitionings will give different estimates.
Leave-one-out Method	A classifier is designed using $(n-1)$ samples and evaluated on the one remaining sample; this is repeated n times with different training sets of size $(n-1)$.	Estimate is unbiased but it has a large variance; large computational requirement because n different classifiers have to be designed.
Rotation Method, n-fold cross validation	A compromise between holdout and leave-one-out methods; divide the available samples into P disjoint subsets, $1 \leq P \leq n$. Use $(P-1)$ subsets for training and the remaining subset for test.	Estimate has lower bias than the holdout method and is cheaper to implement than leave-one-out method.
Bootstrap Method	Generate many bootstrap sample sets of size n by sampling with replacement; several estimators of the error rate can be defined (e.g., E0 and E632) using the bootstrap samples [48].	Bootstrap estimates can have lower variance than the leave-one-out method; computationally more demanding; useful in small sample size situations.

Fig. 9 shows the classification error of the Bayes plug-in linear classifier on the digit dataset as a function of the number of training patterns. The test set error gradually approaches the training set error (resubstitution error) as the number of training samples increases. The relatively large difference between these two error rates for 100 training patterns per class indicates that the bias in these two error estimates can be further reduced by enlarging the training set. Both the curves in this figure represent the average of 50 experiments in which training sets of the given size are randomly drawn; the test set of 1000 patterns is fixed.

The holdout, leave-one-out, and rotation methods are versions of the cross-validation

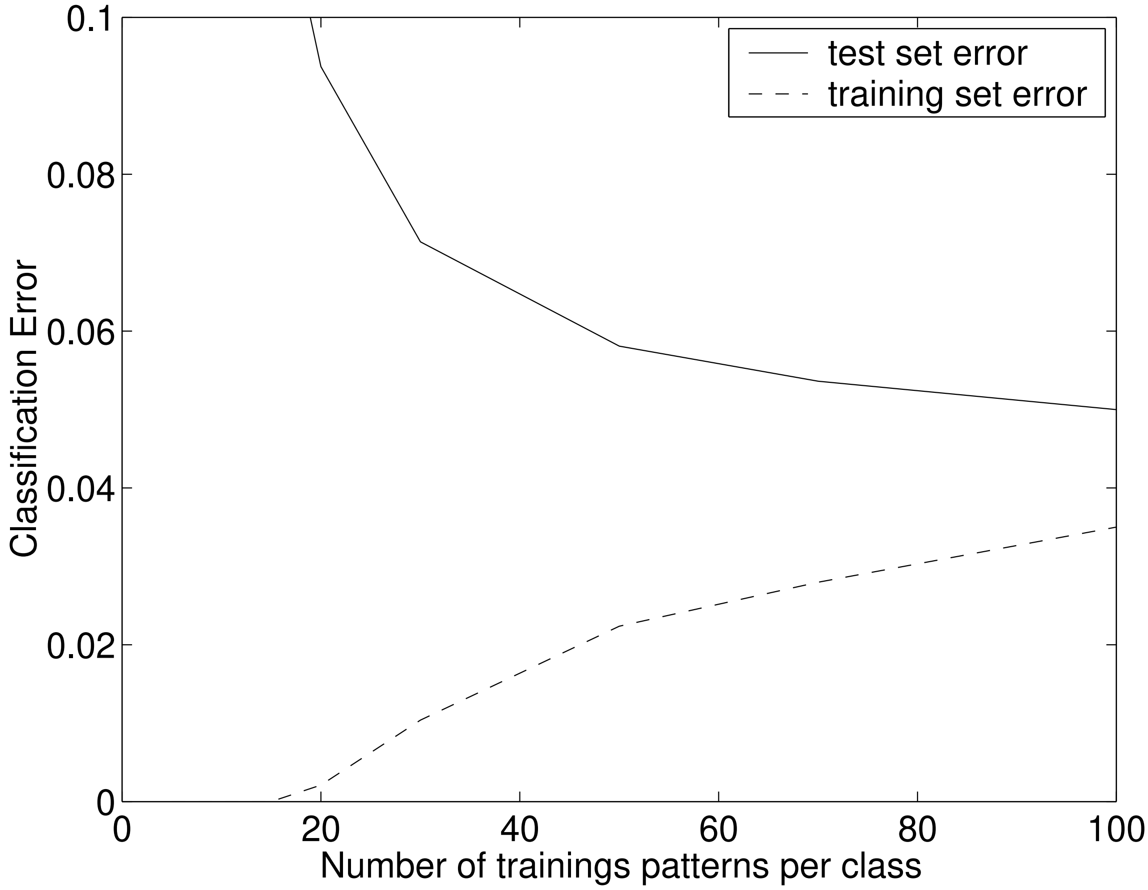


Figure 9: Classification error of the Bayes plug-in linear classifier (equal covariance matrices) as a function of the number of training samples (learning curve) for the test set and the training set on the digit dataset.

approach. One of the main disadvantages of cross-validation methods, especially for small sample size situations, is that not all the available n samples are used for training the classifier. Further, the two extreme cases of cross validation, hold out method and leave-one-out method, suffer from either large bias or large variance, respectively. To overcome this limitation, the bootstrap method [48] has been proposed to estimate the error rate. The bootstrap method resamples the available patterns with replacement to generate a number of "fake" data sets (typically, several hundred) of the same size as the given training set. These new training sets can be used not only to estimate the bias of the resubstitution estimate, but also to define other, so called bootstrap estimates of the error rate. Experimental results have shown that the bootstrap estimates can outperform the cross validation estimates and the resubstitution estimates of the error rate [82].

In many pattern recognition applications, it is not adequate to characterize the performance of a classifier by a single number, $\hat{P}_e$, which measures the overall error rate of a system. Consider the problem of evaluating a fingerprint matching system, where two different yet related error rates are of interest. The False Acceptance Rate (FAR) is the ratio of the number of pairs of different fingerprints that are incorrectly matched by a given system to the total number of match attempts. The False Reject Rate (FRR) is the ratio of the number of pairs of the same fingerprint that are not matched by a given system to the total number of match attempts. A fingerprint matching system can be tuned (by setting an appropriate threshold on the matching score) to operate at a desired value of FAR. However, if we try to decrease the FAR of the system, then it would increase the FRR and vice versa. The Receiver Operating Characteristic (ROC) Curve [107] is a plot of FAR versus FRR which permits the system designer to assess the performance of the recognition system at various operating points (thresholds in the decision rule). In this sense, ROC provides a more comprehensive performance measure than, say, the equal error rate of the system (where FRR = FAR). Fig. 10 shows the ROC curve for the digit dataset where the Bayes plug-in linear classifier is trained on 100 patterns per class. Examples of the use of ROC analysis are combining classifiers [170], and feature selection [99].

In addition to the error rate, another useful performance measure of a classifier is its reject rate. Suppose a test pattern falls near the decision boundary between the two classes. While the decision rule may be able to correctly classify such a pattern, this classification will be made with a low confidence. A better alternative would be to reject

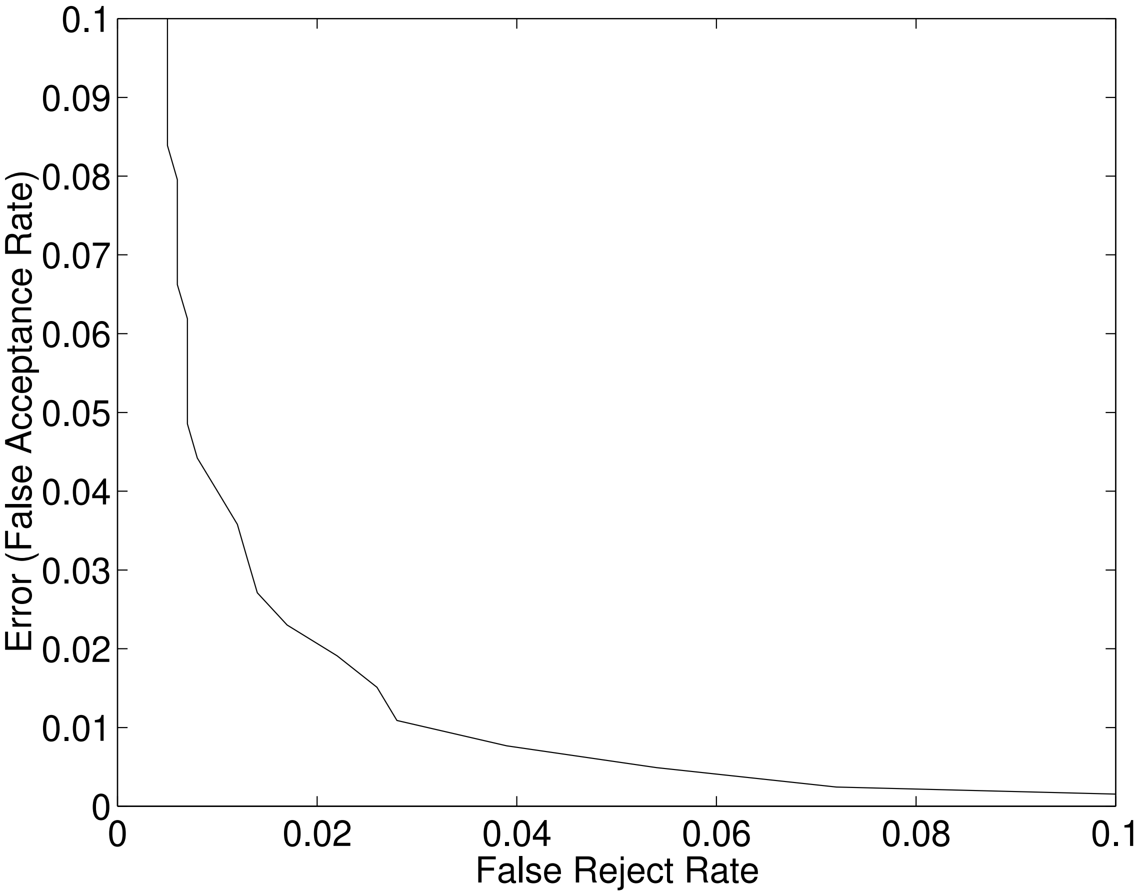


Figure 10: The ROC curve of the Bayes plug-in linear classifier for the digit dataset.

these doubtful patterns instead of assigning them to one of the categories under consideration. How do we decide when to reject a test pattern? For the Bayes decision rule, a well-known reject option is to reject a pattern if its maximum *a posteriori* probability is below a threshold; the larger the threshold, the higher the reject rate. Invoking the reject option reduces the error rate; the larger the reject rate, the smaller the error rate. This relationship is represented as an error-reject trade-off curve which can be used to set the desired operating point of the classifier. Fig. 11 shows the error-reject curve for the digit dataset when a Bayes plug-in linear classifier is used. This curve is monotonically non-increasing, since rejecting more patterns either reduces the error rate or keeps it the same. A good choice for the reject rate is based on the costs associated with reject and incorrect decisions (See [66] for an applied example of the use of error-reject curves).

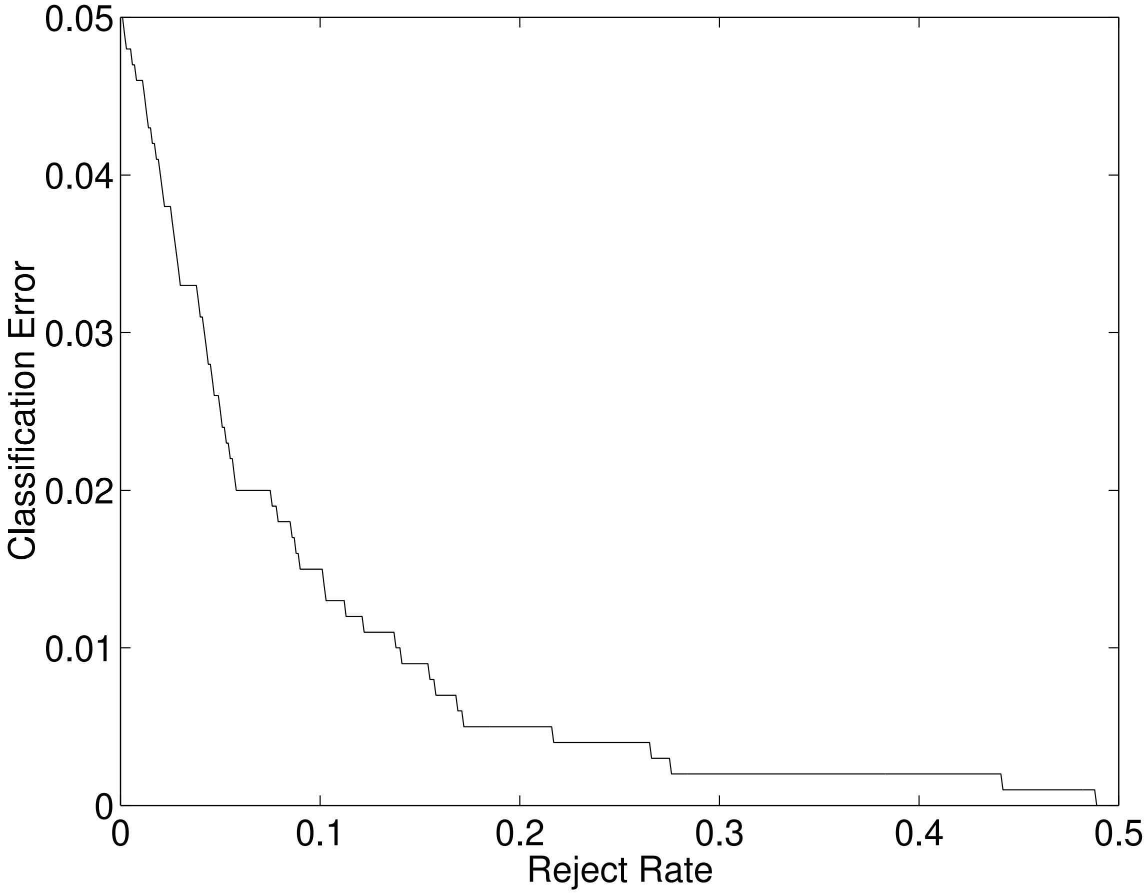


Figure 11: Error-reject curve of the Bayes plug-in linear classifier for the digit dataset.

8 Unsupervised Classification

In many applications of pattern recognition, it is extremely difficult or expensive, or even impossible, to reliably label a training sample with its true category. Consider, for example, the application of land-use classification in remote sensing. In order to obtain the "ground truth" information (category for each pixel) in the image, either the specific site associated with the pixel should be visited or its category should be extracted from a Geographical Information System, if one is available. Unsupervised classification refers to situations where the objective is to construct decision boundaries based on unlabeled training data. Unsupervised classification is also known as data clustering which is a generic label for a variety of procedures designed to find natural groupings, or clusters, in multi-dimensional data, based on measured or perceived similarities among the patterns [81]. Category labels and other information about the source of the data influence the interpretation of the clustering, not the formation of the clusters.

Unsupervised classification or clustering is a very difficult problem because data can reveal clusters with different shapes and sizes (see Fig. 12). To compound the problem

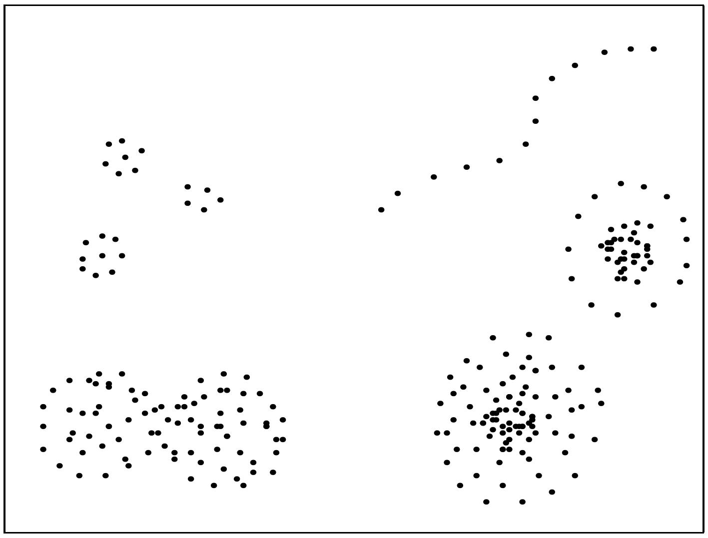


Figure 12: Clusters with different shapes and sizes.

further, the number of clusters in the data often depends on the resolution (fine vs. coarse) with which we view the data. One example of clustering is the detection and delineation of a region containing a high density of patterns compared to the background. A number of functional definitions of a cluster have been proposed which include: (i) patterns within a cluster are more similar to each other than are patterns belonging to different clusters, and (ii) a cluster consists of a relatively high density of points separated from other clusters by a relatively low density of points. Even with these functional definitions of a cluster, it is not easy to come up with an operational definition of clusters. One of the challenges is to select an appropriate measure of similarity to define clusters which, in general, is both data (cluster shape) and context dependent.

Cluster analysis is a very important and useful technique. The speed, reliability, and consistency with which a clustering algorithm can organize large amounts of data constitute overwhelming reasons to use it in applications such as data mining [88], information retrieval [17, 25], image segmentation [55], signal compression and coding [1], and machine learning [25]. As a consequence, hundreds of clustering algorithms have been proposed in the literature and new clustering algorithms continue to appear. However, most of these algorithms are based on the following two popular clustering techniques: iterative square-error partitional clustering and agglomerative hierarchical clustering. Hierarchical techniques organize data in a nested sequence of groups which can be displayed in the form of a dendrogram or a tree. Square-error partitional algorithms attempt to obtain that partition which minimizes the within-cluster scatter or maximizes the between-cluster scatter. To guarantee that an optimum solution has been obtained, one has to examine all possible partitions of the n d-dimensional patterns into K clusters (for a given K), which is not computationally feasible. So various heuristics are used to reduce the search, but then there is no guarantee of optimality.

Partitional clustering techniques are used more frequently than hierarchical techniques in pattern recognition applications, so we will restrict our coverage to partitional methods. Recent studies in cluster analysis suggest that a user of a clustering algorithm should keep the following issues in mind: (i) every clustering algorithm will find clusters in a given dataset whether they exist or not; the data should, therefore, be subjected to tests for clustering tendency before applying a clustering algorithm, followed by a validation of the clusters generated by the algorithm; (ii) there is no "best" clustering algorithm. Therefore, a user is advised to try several clustering algorithms on a given dataset. Further, issues of data collection, data representation, normalization, and cluster validity are as important as the choice of clustering strategy.

The problem of partitional clustering can be formally stated as follows. Given n patterns in a d-dimensional metric space, determine a partition of the patterns into K clusters, such that the patterns in a cluster are more similar to each other than to patterns in different clusters [81]. The value of K may or may not be specified. A clustering criterion, either global or local, must be adopted. A *global* criterion, such as square-error, represents each cluster by a prototype and assigns the patterns to clusters according to the most similar prototypes. A *local* criterion forms clusters by utilizing local structure in the data. For example, clusters can be formed by identifying high-density regions in the pattern space or by assigning a pattern and its k nearest neighbors to the same cluster.

Most of the partitional clustering techniques implicitly assume continuous-valued feature vectors so that the patterns can be viewed as being embedded in a metric space. If the features are on a nominal or ordinal scale, Euclidean distances and cluster centers are not very meaningful, so hierarchical clustering methods are normally applied. Wong and Wang [169] proposed a clustering algorithm for discrete-valued data. The technique of conceptual clustering or learning from examples [108] can be used with patterns represented by nonnumeric or symbolic descriptors. The objective here is to group patterns into conceptually simple classes. Concepts are defined in terms of attributes and patterns are arranged into a hierarchy of classes described by concepts.

In the following subsections, we briefly summarize the two most popular approaches to partitional clustering: square-error clustering and mixture decomposition. A square-error clustering method can be viewed as a particular case of mixture decomposition. We should also point out the difference between a clustering criterion and a clustering algorithm. A clustering algorithm is a particular implementation of a clustering criterion. In this sense, there are a large number of square-error clustering algorithms, each minimizing the square-error criterion and differing from the others in the choice of the algorithmic parameters. Some of the well-known clustering algorithms are listed in Table 10 [81].

Table 10: Clustering Algorithms

Algorithm	Property	Comments
K-means	Identifies hyperspherical clusters; could be modified to find hyper-ellipsoidal clusters using Mahalanobis distance; computationally efficient.	Need to specify K and the initial cluster centers. Additional parameters for creating new clusters, merging existing clusters and outlier detection can be provided.
Fuzzy K-means	Similar to K-means except that every pattern has a degree of membership into the K clusters (fuzzy partition).	Need to specify K, initial cluster centers and cluster membership function.
Minimum Spanning Tree (MST)	Clusters are formed by deleting inconsistent edges in the MST of the data.	Need to provide the definition of an inconsistent edge.
Mutual Neighborhood	Compute the mutual neighborhood value (MNV) for every pair of patterns. If x_j is the p^{th} near neighbor of x_i and x_i is the q^{th} near neighbor of x_j, then $MNV(x_i, x_j) = p + q$; $p, q = 1, \cdots, K$.	Need to specify the neighborhood depth, K.
Single-Link (SL)	A hierarchical clustering algorithm which accepts a $n \times n$ proximity matrix; output is a dendrogram or a tree structure; a single-link cluster is a maximally connected subgraph on the patterns.	Single-link clusters easily chain together and are often "straggly"; need a heuristic to cut the tree to form clusters (a partition).
Complete-Link (CL)	A hierarchical clustering algorithm which accepts a $n \times n$ proximity matrix; output is a dendrogram or a tree structure; a complete-link cluster is a maximally complete subgraph on the patterns.	Complete-link clusters tend to be small and compact which combine nicely into layer clusters even when such a hierarchy is not warranted; need a heuristic to form clusters (a partition).
Mixture Decomposition	Each pattern is assumed to be drawn from one of K underlying populations, or clusters; population parameters are estimated from unlabelled data.	The form and the number of underlying population (K) densities are assumed to be known; K can be estimated using a number of criteria (see Section 8.2).

8.1 Square-Error Clustering

The most commonly used partitional clustering strategy is based on the square-error criterion. The general objective is to obtain that partition which, for a fixed number of clusters, minimizes the square-error. Suppose that the given set of n patterns in d dimensions has somehow been partitioned into K clusters $\{C_1, C_2, \cdots, C_k\}$ such that cluster C_k has n_k patterns and each pattern is in exactly one cluster, so that $\sum_{k=1}^{K} n_k = n$.

The mean vector, or center, of cluster C_k is defined as the centroid of the cluster, or

$$\boldsymbol{m}^{(k)} = \left(\frac{1}{n_k}\right) \sum_{i=1}^{n_k} \boldsymbol{x}_i^{(k)}, \tag{14}$$

where $\boldsymbol{x}_i^{(k)}$ is the i^{th} pattern belonging to cluster C_k. The square-error for cluster C_k is the sum of the squared Euclidean distances between each pattern in C_k and its cluster center $\boldsymbol{m}^{(k)}$. This square-error is also called the within-cluster variation

$$e_k^2 = \sum_{i=1}^{n_k} \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{m}^{(k)}\right)^T \left(\boldsymbol{x}_i^{(k)} - \boldsymbol{m}^{(k)}\right). \tag{15}$$

The square-error for the entire clustering containing K clusters is the sum of the within-cluster variations:

$$E_K^2 = \sum_{k=1}^{K} e_k^2. \tag{16}$$

The objective of a square-error clustering method is to find a partition containing K clusters that minimizes E_K^2 for a fixed K. The resulting partition has also been referred to as the minimum variance partition. A general algorithm for the iterative partitional clustering method is given below.

ALGORITHM FOR ITERATIVE PARTITIONAL CLUSTERING

Step 1. Select an initial partition with K clusters. Repeat steps 2 through 5 until the cluster membership stabilizes.

Step 2. Generate a new partition by assigning each pattern to its closest cluster center.

Step 3. Compute new cluster centers as the centroids of the clusters.

Step 4. Repeat steps 2 and 3 until an optimum value of the criterion function is found.

Step 5. Adjust the number of clusters by merging and splitting existing clusters or by removing small, or outlier, clusters.

The above algorithm, without step 5, is also known as the K-means algorithm. The details of the steps in this algorithm must either be supplied by the user as parameters or be implicitly hidden in the computer program. However, these details are crucial to the success of the program. A big frustration in using clustering programs is the lack of guidelines available for choosing K, initial partition, updating the partition, adjusting the number of clusters, and the stopping criterion [8].

The simple K-means partitional clustering algorithm described above is computationally efficient and gives surprisingly good results if the clusters are compact, hyperspherical in shape and well-separated in the feature space. If the Mahalanobis distance is used in defining the squared error in Eq. (16), then the algorithm is even able to detect hyperellipsoidal shaped clusters. Numerous attempts have been made to improve the performance of the basic K-means algorithm by (i) incorporating a fuzzy criterion function

[15], resulting in a fuzzy K-means (or c-means) algorithm, (ii) using genetic algorithms, simulated annealing, deterministic annealing, and tabu search to optimize the resulting partition [110], [139], and (iii) mapping it onto a neural network [103] for possibly efficient implementation. However, many of these so-called enhancements to the K-means algorithm are computationally demanding and require additional user-specified parameters for which no general guidelines are available. Judd et al. [88] show that a combination of algorithmic enhancements to a square-error clustering algorithm and distribution of the computations over a network of workstations can be used to cluster hundreds of thousands of multi-dimensional patterns in just a few minutes.

It is interesting to note how seemingly different concepts for partitional clustering can lead to essentially the same algorithm. It is easy to verify that the generalized Lloyd vector quantization algorithm used in the communication and compression domain is equivalent to the K-means algorithm. A vector quantizer (VQ) is described as a combination of an encoder and a decoder. A d-dimensional VQ consists of two mappings: an encoder γ which maps the input alphabet ($\mathbf{A}$) to the channel symbol set ($\mathbf{M}$), and a decoder β which maps the channel symbol set ($\mathbf{M}$) to the output alphabet ($\hat{\mathbf{A}}$), i.e., $\gamma(\boldsymbol{y}) : \mathbf{A} \rightarrow \mathbf{M}$ and $\beta(\boldsymbol{v}) : \mathbf{M} \rightarrow \hat{\mathbf{A}}$. A distortion measure $\mathcal{D}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ specifies the cost associated with quantization, where $\hat{\boldsymbol{y}} = \beta(\gamma(\boldsymbol{y}))$. Usually, an optimal quantizer minimizes the average distortion under a size constraint on $\mathbf{M}$. Thus, the problem of vector quantization can be posed as a clustering problem, where the number of clusters K is now the size of the output alphabet, $\hat{\mathbf{A}} : \{\hat{\boldsymbol{y}}_{\boldsymbol{i}}, i = 1, \ldots, K\}$, and the goal is to find a quantization (referred to as a partition in the K-means algorithm) of the d-dimensional feature space which minimizes the average distortion (mean square error) of the input patterns. Vector quantization has been widely used in a number of compression and coding applications, such as speech waveform coding, image coding, etc., where only the symbols for the output alphabet or the cluster centers are transmitted instead of the entire signal [67], [32]. Vector quantization also provides an efficient tool for density estimation [68]. A kernel-based approach (e.g., a mixture of Gaussian kernels, where each kernel is placed at a cluster center) can be used to estimate the probability density of the training samples. A major issue in VQ is the selection of the output alphabet size. A number of techniques, such as the minimum description length (MDL) principle [138], can be used to select this parameter (see Section 8.2). The supervised version of VQ is called learning vector quantization (LVQ) [92].

8.2 Mixture Decomposition

Finite mixtures are a flexible and powerful probabilistic modeling tool. In statistical pattern recognition, the main use of mixtures is in defining formal (i.e., model-based) approaches to unsupervised classification [81]. The reason behind this is that mixtures adequately model situations where each pattern has been produced by one of a set of alternative (probabilistically modeled) sources [155]. Nevertheless, it should be kept in mind that strict adherence to this interpretation is not required: mixtures can also be seen as a class of models that are able to represent arbitrarily complex probability density functions. This makes mixtures also well suited for representing complex class-conditional densities in supervised learning scenarios (see [137] and references therein). Finite mixtures can also be used as a feature selection tool [127].

8.2.1 Basic Definitions

Consider the following scheme for generating random samples. There are K random sources, each characterized by a probability (mass or density) function $p_m(\mathbf{y}|\boldsymbol{\theta}_m)$, parameterized by $\boldsymbol{\theta}_m$, for $m = 1, ..., K$. Each time a sample is to be generated, we randomly choose one of these sources, with probabilities $\{\alpha_1, ..., \alpha_K\}$, and then sample from the chosen source. The random variable defined by this (two-stage) compound generating mechanism is characterized by a finite mixture distribution; formally, its probability function is

$$p(\mathbf{y}|\boldsymbol{\Theta}_{(K)}) = \sum_{m=1}^{K} \alpha_m p_m(\mathbf{y}|\boldsymbol{\theta}_m), \tag{17}$$

where each $p_m(\mathbf{y}|\boldsymbol{\theta}_m)$ is called a component, and $\boldsymbol{\Theta}_{(K)} = \{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_K, \alpha_1, ..., \alpha_{K-1}\}$. It is usually assumed that all the components have the same functional form; for example, they are all multivariate Gaussian. Fitting a mixture model to a set of observations $\mathbf{y} = \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(n)}\}$ consists of estimating the set of mixture parameters that best describe this data. Although mixtures can be built from many different types of components, the majority of the literature focuses on Gaussian mixtures [155].

The two fundamental issues arising in mixture fitting are: (i) how to estimate the parameters defining the mixture model, and (ii) how to estimate the number of components [159]. For the first question, the standard answer is the *expectation-maximization* (EM) algorithm (which, under mild conditions, converges to the maximum likelihood (ML) estimate of the mixture parameters); several authors have also advocated the (computationally demanding) *Markov chain Monte-Carlo* (MCMC) method [135]. The second question is more difficult; several techniques have been proposed which are summarized in Section 8.2.3. Note that the output of the mixture decomposition is as good as the validity of the assumed component distributions.

8.2.2 EM Algorithm

The expectation-maximization algorithm interprets the given observations $\mathbf{y}$ as *incomplete* data, with the missing part being a set of labels associated with $\mathbf{y}$, $\mathbf{z} = \{\mathbf{z}^{(1)}, ..., \mathbf{z}^{(K)}\}$. Missing variable $\mathbf{z}^{(i)} = [z_1^{(i)}, ..., z_K^{(i)}]^T$ indicates which of the K components generated $\mathbf{y}^{(i)}$; if it was the m-th component, then $z_m^{(i)} = 1$ and $z_p^{(i)} = 0$, for $p \neq m$ [155]. In the presence of both $\mathbf{y}$ and $\mathbf{z}$, the (complete) log-likelihood can be written as

$$L_c\left(\boldsymbol{\Theta}_{(K)}, \mathbf{y}, \mathbf{z}\right) = \sum_{j=1}^{n}\sum_{m=1}^{K} z_m^{(j)} \log\left[\alpha_m p_m(\mathbf{y}^{(j)}|\boldsymbol{\theta}_m)\right], \quad \sum_{i=1}^{K}\alpha_i = 1. \tag{18}$$

The EM algorithm proceeds by alternatively applying the following two steps:

- **E-step:** Compute the conditional expectation of the complete log-likelihood (given $\mathbf{y}$ and the current parameter estimate, $\widehat{\boldsymbol{\Theta}}_{(K)}^{(t)}$). Since Eq. (18) is linear in the missing variables, the E-step for mixtures reduces to the computation of the conditional expectation of the missing variables: $w_m^{(i,t)} \equiv E[z_m^{(i)}|\widehat{\boldsymbol{\Theta}}_{(K)}^{(t)}, \mathbf{y}]$.
- **M-step:** Update the parameter estimates: $\widehat{\boldsymbol{\Theta}}_{(K)}^{(t+1)} = \arg\max_{\boldsymbol{\Theta}_{(K)}} Q(\boldsymbol{\Theta}_{(K)}, \widehat{\boldsymbol{\Theta}}_{(K)}^{(t)})$. For the mixing probabilities, this becomes

$$\widehat{\alpha}_m^{(t+1)} = \frac{1}{n}\sum_{i=1}^{n} w_m^{(i,t)}, m = 1, 2, \cdots, K-1. \tag{19}$$

In the Gaussian case, each $\boldsymbol{\theta}_m$ consists of a mean vector and a covariance matrix which are updated using weighted versions (with weights $\widehat{\alpha}_m^{(t+1)}$) of the standard ML estimates [155].

The main difficulties in using EM for mixture model fitting, which are current research topics, are: its local nature, which makes it critically dependent on initialization; the possibility of convergence to a point on the boundary of the parameter space with unbounded likelihood (i.e., one of the α_m approaches zero with the corresponding covariance becoming arbitrarily close to singular).

8.2.3 Estimating the Number of Components

The ML criterion can not be used to estimate the number of mixture components because the maximized likelihood is a non-decreasing function of K, thereby making it useless as a model selection criterion (selecting a value for K in this case). This is a particular instance of the *identifiability* problem where the classical (χ^2-based) hypothesis testing can not be used because the necessary regularity conditions are not met [155]. Several alternative approaches that have been proposed are summarized below.

EM-based approaches use the (fixed K) EM algorithm to obtain a sequence of parameter estimates for a range of values of K, $\{\widehat{\mathbf{\Theta}}_{(K)},\ K = K_{\min}, ..., K_{\max}\}$; the estimate of K is then defined as the minimizer of some cost function,

$$\widehat{K} = \arg\min_K \left\{\mathcal{C}\left(\widehat{\mathbf{\Theta}}_{(K)}, K\right),\ K = K_{\min}, ..., K_{\max}\right\}. \tag{20}$$

Most often, this cost function includes the maximized log-likelihood function plus an additional term whose role is to penalize large values of K. An obvious choice in this class is to use the minimum description length (MDL) criterion [10][138], but several other model selection criteria have been proposed: Schwarz's Bayesian inference criterion (BIC), the minimum message length (MML) criterion, and Akaike's information criterion

(AIC) [2], [148], [167].

Resampling-based schemes and cross-validation-type approaches have also been suggested; these techniques are (computationally) much closer to stochastic algorithms than to the methods in the previous paragraph. Stochastic approaches generally involve Markov chain Monte Carlo (MCMC) [135] sampling and are far more computationally intensive than EM. MCMC has been used in two different ways: to implement model selection criteria to actually estimate K; and, with a more "fully Bayesian flavor", to sample from the full *a posteriori* distribution where K is included as an unknown. Despite their formal appeal, we think that MCMC-based techniques are still far too computationally demanding to be useful in pattern recognition applications.

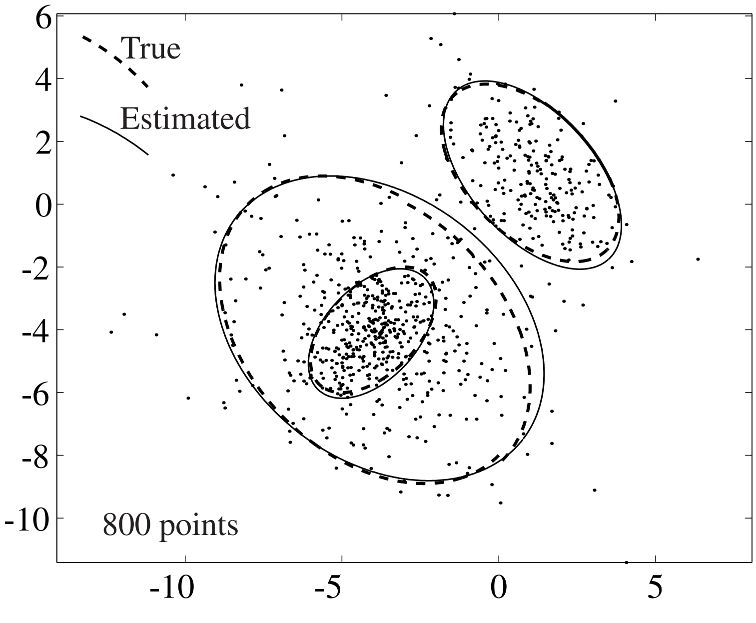


Figure 13: Mixture Decomposition Example.

Fig. 13 shows an example of mixture decomposition, where K is selected using a modified MDL criterion [51]. The data consists of 800 two-dimensional patterns distributed over three Gaussian components; two of the components have the same mean vector but different covariance matrices and that is why one dense cloud of points is inside another cloud of rather sparse points. The level curve contours (of constant Mahalanobis distance) for the true underlying mixture and the estimated mixture are superimposed on the data. For details, see [51]. Note that a clustering algorithm such as K-means will not be able to identify these three components, due to the substantial overlap of two of these components.

9 Discussion

In its early stage of development, statistical pattern recognition focused mainly on the core of the discipline: the Bayesian decision rule and its various derivatives (such as linear and quadratic discriminant functions), density estimation, the curse of dimensionality problem, and error estimation. Due to the limited computing power available in the 1960s and 1970s, statistical pattern recognition employed relatively simple techniques which were applied to small-scale problems.

Since the early 1980s, statistical pattern recognition has experienced a rapid growth. Its frontiers have been expanding in many directions simultaneously. This rapid expansion is largely driven by the following forces.

1. Increasing interaction and collaboration among different disciplines, including neural networks, machine learning, statistics, mathematics, computer science, and biology. These multi-disciplinary efforts have fostered new ideas, methodologies, and techniques which enrich the traditional statistical pattern recognition paradigm.

2. The prevalence of fast processors, the Internet, large and inexpensive memory and storage. The advanced computer technology has made it possible to implement complex learning, searching and optimization algorithms which was not feasible a

few decades ago. It also allows us to tackle large-scale real world pattern recognition problems which may involve millions of samples in high dimensional spaces (tens of thousands of features).

3. Emerging applications, such as data mining and document taxonomy creation and maintenance. These emerging applications have brought new challenges that foster a renewed interest in statistical pattern recognition research.

4. Last but not the least, the need for a principled, rather than *ad hoc*, approach for successfully solving pattern recognition problems in a predictable way. For example, many concepts in neural networks, which were inspired by biological neural networks, can be directly treated in a principled way in statistical pattern recognition.

9.1 Frontiers of Pattern Recognition

Table 11 summarizes several topics which, in our opinion, are at the frontiers of pattern recognition. As we can see from Table 11, many fundamental research problems in statistical pattern recognition remain at the forefront even as the field continues to grow. One such example, model selection (which is an important issue in avoiding the curse of dimensionality), has been a topic of continued research interest. A common practice in model selection relies on cross-validation (rotation method), where the best model is selected based on the performance on the validation set. Since the validation set is not used in training, this method does not fully utilize the precious data for training which is especially undesirable when the training data set is small. To avoid this problem, a number of model selection schemes [71] have been proposed, including Bayesian methods

[14], minimum description length (MDL) [138], Akaike information criterion (AIC) [2] and marginalized likelihood [101, 159]. Various other regularization schemes which incorporate prior knowledge about model structure and parameters have also been proposed. Structural risk minimization based on the notion of VC dimension has also been used for model selection where the best model is the one with the best worst-case performance (upper bound on the generalization error) [162]. However, these methods do not reduce the complexity of the search for the best model. Typically, the complexity measure has to be evaluated for every possible model or in a set of pre-specified models. Certain assumptions (e.g., parameter independence) are often made in order to simplify the complexity evaluation. Model selection based on stochastic complexity has been applied to feature selection in both supervised learning and unsupervised learning [159] and pruning in decision trees [106]. In the latter case, the best number of clusters is also automatically determined.

Another example is mixture modeling using EM algorithm (see Section 8.2), which was proposed in 1977 [36], and which is now a very popular approach for density estimation and clustering [159], due to the computing power available today.

Over the recent years, a number of new concepts and techniques have also been introduced. For example, the maximum margin objective was introduced in the context of support vector machines [23] based on structural risk minimization theory [162]. A classifier with a large margin separating two classes has a small VC dimension, which yields a good generalization performance. Many successful applications of SVMs have demonstrated the superiority of this objective function over others [72]. It is found that the boosting algorithm [143] also improves the margin distribution. The maximum margin objective can be considered as a special regularized cost function, where the regularizer is the inverse of the margin between the two classes. Other regularized cost functions, such as weight decay and weight elimination, have also been used in the context of neural networks.

Due to the introduction of SVMs, linear and quadratic programming optimization techniques are once again being extensively studied for pattern classification. Quadratic programming is credited for leading to the nice property that the decision boundary is fully specified by boundary patterns, while linear programming with the L^1 norm or the inverse of the margin yields a small set of features when the optimal solution is obtained.

The topic of local decision boundary learning has also received a lot of attention. Its primary emphasis is on using patterns near the boundary of different classes to construct or modify the decision boundary. One such an example is the boosting algorithm and its variation (AdaBoost) where misclassified patterns, mostly near the decision boundary, are subsampled with higher probabilities than correctly classified patterns to form a new training set for training subsequent classifiers. Combination of local experts is also related to this concept, since local experts can learn local decision boundaries more accurately than global methods. In general, classifier combination could refine decision boundary such that its variance with respect to Bayes decision boundary is reduced, leading to improved recognition accuracy [158].

Sequential data arise in many real world problems, such as speech and on-line handwriting. Sequential pattern recognition has, therefore, become a very important topic in pattern recognition. Hidden Markov Models (HMM), have been a popular statistical tool for modeling and recognizing sequential data, in particular, speech data [130], [86]. A large number of variations and enhancements of HMMs have been proposed in the literature [12], including hybrids of HMMs and neural networks, input-output HMMs, weighted transducers, variable-duration HMMs, Markov switching models, and switching state-space models.

The growth in sensor technology and computing power has enriched the availability of data in several ways. Real world objects can now be represented by many more measurements and sampled at high rates. As physical objects have a finite complexity, these measurements are generally highly correlated. This explains why models using spatial and spectral correlation in images, or the Markov structure in speech, or subspace approaches in general, have become so important; they compress the data to what is physically meaningful, thereby improving the classification accuracy simultaneously.

Supervised learning requires that every training sample be labeled with its true category. Collecting a large amount of labeled data can sometimes be very expensive. In practice, we often have a small amount of labeled data and a large amount of unlabeled data. How to make use of unlabeled data for training a classifier is an important problem. SVM has been extended to perform semi-supervised learning [13].

Invariant pattern recognition is desirable in many applications, such as character and face recognition. Early research in statistical pattern recognition did emphasize extraction of invariant features which turns out to be a very difficult task. Recently, there has been some activity in designing invariant recognition methods which do not require invariant features. Examples are the nearest neighbor classifier using tangent distance [152] and deformable template matching [84].These approaches only achieve invariance to small amounts of linear transformations and nonlinear deformations. Besides, they are computationally very intensive. Simard et al. [153] proposed an algorithm named Tangent-Prop to minimize the derivative of the classifier outputs with respect to distortion parameters, i.e., to improve the invariance property of the classifier to the selected distortion. This makes the trained classifier computationally very efficient.

9.2 Concluding Remarks

Watanabe [164] wrote in the preface of his 1972 book entitled "Frontiers of Pattern Recognition," that "Pattern recognition is a fast-moving and proliferating discipline. It is not easy to form a well-balanced and well-informed summary view of the newest developments in this field. It is still harder to have a vision of its future progress."

Table 11: FRONTIERS OF PATTERN RECOGNITION

Topic	Examples	Comments
Model selection and generalization	Bayesian learning, MDL, AIC, marginalized likelihood, structural risk.	Make full use of the available data for training.
Mixture modeling and EM algorithm	Clustering density estimation.	Soft membership; better than k-means clustering.
New objective functions for classification	Maximum margin (SVMs), regularized cost.	Provide low VC dimension and good generalization.
Optimization methods	Quadratic programming; linear programming.	Leads to support vectors; built-in feature selection.
Local decision boundary learning	SVMs, Boosting, mixture of local experts.	Focus on boundary patterns.
Sequential pattern recognition	Hidden Markov Models (HMMs), recurrent networks.	Successfully applied to speech and handwriting recognition.
Local-invariant (dis)similarity measures	Deformable template matching, tangent distance.	Invariant to local distortions.
Independent component analysis	Blind source separation, feature extraction.	Extract statistically independent components.
Combining multiple classifiers	See Table 7.	Improve recognition accuracy.
Emerging applications	Data mining and KDD, Document categorization, Image database retrieval, Financial forecasting, Biometric recognition (fingerprint, iris, face, voice, handwriting and signature).	Large volume, high dimension, mixed data types, missing data, data modeling, model selection.

// 10 Acknowledgment

The authors would like to thank the anonymous reviewers, and Drs. Roz Picard, Tin Ho, and Mario Figueiredo for their critical and constructive comments and suggestions. The authors are indebted to Dr. Kevin Bowyer for his careful review and to his students for their valuable feedback. The authors are also indebted to following colleagues for their valuable assistance in preparing this manuscript: Elzbieta Pekalska, Dick de Ridder, Scott Connell, Aditya Vailaya, Vincent Hsu, Dr. Pavel Pudil, and Dr. Patrick Flynn.

References

[1] Abbas and M.M. Fahmy, "Neural networks for maximum likelihood clustering," *Signal Processing*, vol. 36, no. 1, pp. 111-126, 1994.

[2] H. Akaike, "A new look at statistical model identification," *IEEE Trans. on Automatic Control*, vol. AC-19, pp. 716-723, 1974.

[3] S. Amari, T. P. Chen, and A. Cichocki, "Stability analysis of learning algorithms for blind source separation," *Neural Networks*, vol. 10, no. 8, pp. 1345-1351, 1997.

[4] J. A. Anderson, "Logistic discrimination," in *Handbook of Statistics*, P. R. Krishnaiah and L. N. Kanal, eds., vol. 2, pp. 169-191, North Holland, Amsterdam, 1982.

[5] J. Anderson, A. Pellionisz, and E. Rosenfeld, *Neurocomputing 2: Directions for Research*, MIT Press, Cambridge CA, 1990.

[6] A. Antos, L. Devroye, and L. Gyorfi, "Lower Bounds for Bayes Error Estimation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 21, no. 7, pp. 643-645, 1999.

[7] H. Avi-Itzhak and T. Diep, "Arbitrarily tight upper and lower bounds on the Bayesian probability of error," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 1, pp. 89-91, 1996.

[8] E. Backer, *Computer-Assisted Reasoning in Cluster Analysis*, Prentice Hall, 1995.

[9] R. Bajcsy and S. Kovacic, "Multiresolution elastic matching," *Comput. Vision Graphics Image Process.*, vol. 46, pp. 1-21, 1989.

[10] A. Barron and J. Rissanen and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Trans. on Information Theory*, vol. 44, no. 6, pp. 2743-2760, Oct. 1998.

[11] A. Bell and T. Sejnowski, "An information-maximization approach to blind separation," *Neural Computation*, vol. 7, pp. 1004-1034, 1995.

[12] Y. Bengio, "Markovian models for sequential data," *Neural Computing Surveys*, vol. 2, pp. 129-162, 1999. <http://www.icsi.berkeley.edu/~jagota/NCS>

[13] K. P. Bennett, "Semi-supervised support vector machines," *Proc. of Neural Information Processing Systems*, Denver, CO, 1998.

[14] J. Bernardo and A. Smith, *Bayesian Theory*, John Wiley & Sons, 1994.

[15] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

[16] J. C. Bezdek and S. K. Pal, Eds., *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*, IEEE Press, 1992.

[17] S.K. Bhatia and J.S. Deogun, "Conceptual clustering in information retrieval," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no. 3, pp. 427-436, 1998.

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

[19] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning [review]," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245-271, 1997.

[20] I. Borg and P. Groenen, *Modern Multidimensional Scaling*, Springer Verlag, Berlin, 1997.

[21] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.

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

[23] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-167, 1998.

[24] J. Cardoso, "Blind signal separation: Statistical principles," *Proc. IEEE*, vol. 86, pp. 2009-2025, 1998.

[25] C. Carpineto and G. Romano, "A lattice conceptual clustering system and its application to browsing retrieval," *Machine Learning*, vol. 24, no. 2, pp. 95-122, 1996.

[26] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feed-forward neural networks," *IEEE Trans. on Neural Networks*, vol. 8, no. 3, pp. 519-531, 1997.

[27] C. Chatterjee and V. P. Roychowdhury, "On self-organizing algorithms and networks for class-separability features," *IEEE Trans. on Neural Networks*, vol. 8, no. 3, pp. 663-678, 1997.

[28] B. Cheng and D. M. Titterington, "Neural networks: A review from statistical perspective," *Statistical Science*, vol. 9, no. 1, pp. 2-54, 1994.

[29] H. Chernoff, "The use of faces to represent points in k-dimensional space graphically," *J. Amer. Statist. Ass.*, vol. 68, pp. 361-368, June 1973.

[30] P. A. Chou, "Optimal partitioning for classification and regression trees," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 340-354, 1991.

[31] P. Comon, "Independent component analysis, a new concept?," *Signal Processing*, vol. 36, no. 3, pp. 287-314, 1994.

[32] P. C. Cosman, K. L. Oehler, E. A. Riskin, and R. M. Gray, "Using vector quantization for image processing," *Proc. IEEE*, vol. 81, pp. 1326-1341, Sept. 1993.

[33] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. on Electronic Computers*, vol. EC-14, pp. 326-334, June 1965.

[34] T. M. Cover, "The best two independent measurements are not the two best," *IEEE Trans. Syst. Man, Cybern.*, SMC-4, pp. 116-117, 1974.

[35] T. M. Cover and J. M. Van Campenhout, "On the possible orderings in the measurement selection problem," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 7, no. 9, pp. 657-661, Sept. 1977.

[36] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the (EM) algorithm," *Journal of the Royal Statistical Society*, vol. 39, pp.1-38, 1977.

[37] H. Demuth, H. and M. Beale, *Neural Network Toolbox for use with Matlab*, version 3, Mathworks, Natick, MA, USA, 1998.

[38] D. De Ridder and R. P. W. Duin, "Sammon's mapping using neural networks: comparison," *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1307-1316, 1997.

[39] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice Hall, London, 1982.

[40] L. Devroye, "Automatic pattern recognition: A study of the probability of error," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 530-543, 1988.

[41] L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer Verlag, Berlin, 1996.

[42] A. Djouadi and E. Bouktache, "A fast algorithm for the nearest-neighbor classifier," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 3, pp. 277-282, 1997.

[43] H. Drucker, C. Cortes, L. D. Jackel, Y. Lecun, and V. Vapnik, "Boosting and other ensemble methods," *Neural Computation*, vol. 6, no. 6, pp. 1289-1301, 1994.

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

[45] R. O. Duda, P. E. Hart and D.G. Stork, *Pattern Classification and Scene Analysis*, second edition, John Wiley & Sons, New York, 1999 (in Press).

[46] R. P. W. Duin, "A note on comparing classifiers," *Pattern Recognition Letters*, vol. 17, no. 5, pp. 529-536, 1996.

[47] R. P. W. Duin, D. De Ridder, and D. M. J. Tax, "Experiments with a featureless approach to pattern recognition," *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1159-1166, 1997.

[48] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*, Philadelphia: SIAM, 1982.

[49] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Knowledge discovery and data mining: Towards a unifying framework," *Proc. of the 2nd Intl. Conf. on Knowledge Discovery and Data Mining*, Portland, Oregon, August 1999.

[50] F. Ferri, P. Pudil, M. Hatef and J. Kittler, "Comparative study of techniques for large scale feature selection," in *Pattern Recognition in Practice IV*, E. Gelsema and L. Kanal, eds., pp. 403-413. Elsevier Science B.V., 1994.

[51] M. Figueiredo, J. Leitao and A. K. Jain, "On fitting mixture models," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, E. Hancock and M. Pellillo, eds., Springer Verlag, 1999 (in press).

[52] Y. Freund and R. Schapire. "Experiments with a new boosting algorithm," *Proc. 13th Int. Conf. on Machine Learning*, 1996.

[53] J. H. Friedman, "Exploratory projection pursuit," *J. Amer. Statist. Ass.*, vol. 82, pp. 249-266, 1987.

[54] J. H. Friedman, "Regularized discriminant analysis," *J. Amer. Statist. Ass.*, vol. 84, pp. 165-175, 1989.

[55] H. Frigui and R. Krishnapuram, "A Robust Competitive Clustering Algorithm With Applications in Computer Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 450-465, 1999.

[56] K. S. Fu, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, 1982.

[57] K. S. Fu, "A step towards unification of syntactic and statistical pattern recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, no. 2, pp. 200-205, March 1983.

[58] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, second edition, Academic Press, New York, 1990.

[59] K. Fukunaga and R. R. Hayes, "Effects of sample size in classifier design," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 8, pp. 873-885, 1989.

[60] K. Fukunaga and R. R. Hayes, "The reduced parzen classifier," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 4, pp. 423-425, 1989.

[61] K. Fukunaga and D. M. Hummels, "Leave-one-out procedures for nonparametric error estimates," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 4, pp. 421-423, 1989.

[62] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: a neural network model for a mechanism of visual pattern recognition," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-13, pp. 826-834, 1983.

[63] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, "An iterative growing and pruning algorithm for classification tree design," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 2, pp. 163-174, 1991.

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

[65] C. Glymour, D. Madigan, D. Pregibon, and P. Smyth, "Statistical Themes and Lessons for Data Mining," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 11-28, 1997.

[66] M. Golfarelli, D. Maio, and D. Maltoni, "On the error-reject trade-off in biometric verification system," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 786-796, 1997.

[67] R. M. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, pp. 4-29, April 1984.

[68] R. M. Gray and R. A. Olshen, "Vector quantization and density estimation," in *SEQUENCES97*, 1997. <http://www-isl.stanford.edu/~gray/compression.html>

[69] U. Grenander, *General Pattern Theory*, Oxford University Press, 1993.

[70] D. J. Hand, "Recent advances in error rate estimation," *Pattern Recognition Letters*, vol. 4, no. 5, pp. 335-346, 1986.

[71] M. H. Hansen and B. Yu, "Model selection and the principle of minimum description length," *Technical report*, Lucent Bell Lab, Murray Hill, NJ 07974, 1998.

[72] M. A. Hearst, "Support vector machines," *IEEE Intelligent Systems*, pp. 18-28, July/August 1998.

[73] S. Haykin, *Neural Networks, a Comprehensive Foundation*, second edition, Prentice-Hall, Englewood Cliffs, 1999.

[74] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, no. 1, pp. 66-75, 1994.

[75] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832-844, 1998.

[76] J. P. Hoffbeck and D. A. Landgrebe, "Covariance matrix estimation and classification with limited training data," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 7, pp. 763-767, 1996.

[77] A. Hyvarinen, "Survey on independent component analysis," *Neural Computing Surveys*, vol. 2, pp. 94-128, 1999. <http://www.icsi.berkeley.edu/~jagota/NCS>

[78] A. Hyvarinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1483-1492, Oct. 1997.

[79] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, pp. 79-87, 1991.

[80] A. K. Jain and B. Chandrasekaran, "Dimensionality and sample size considerations in pattern recognition practice," in *Handbook of Statistics*, P.R. Krishnaiah and L.N. Kanal, eds., vol. 2, pp. 835 - 855, North-Holland, Amsterdam, 1982.

[81] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, 1988.

[82] A. K. Jain, R. C. Dubes, and C.-C. Chen, "Bootstrap Techniques for error estimation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 628-633, 1987.

[83] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *IEEE Computer*, pp. 31-44, March 1996.

[84] A. Jain, Y. Zhong, and S. Lakshmanan, "Object matching using deformable templates," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, March 1996.

[85] A. K. Jain and D. Zongker, "Feature selection: evaluation, application, and small sample performance," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153-158, Feb. 1997.

[86] F. Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, 1998.

[87] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, pp. 181-214, 1994.

[88] D. Judd, P. Mckinley and A. K. Jain, "Large-Scale parallel Data Clustering," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 871-876, 1998.

[89] L. N. Kanal, "Patterns in pattern recognition: 1968-1974," *IEEE Trans. Information Theory*, vol. 20, no. 6, pp. 697-722, 1974.

[90] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226-239, 1998.

[91] R. M. Kleinberg, "Stochastic Discrimination," *Annals of Mathematics and Artificial Intelligence*, vol. 1, pp. 207-239, 1990.

[92] T. Kohonen, *Self-Organizing Maps*, Springer Series in Information Sciences, vol. 30, Berlin, 1995.

[93] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretsky, and T. Leen, eds., vol. 7. Cambridge, MA, MIT Press, 1995.

[94] L. Lam and C. Y. Suen, "Optimal combinations of pattern classifiers," *Pattern Recognition Letters*, vol. 16, no. 9, pp. 945-954, 1995.

[95] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten Zip code recognition," *Neural Computation*, vol. 1, pp. 541-551, 1989.

[96] T. W. Lee, *Independent Component Analysis*, Kluwer Academic Publishers, Dordrech, 1998.

[97] C. Lee and D.A. Landgrebe, "Feature extraction based on decision boundaries," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 15, no. 4, pp. 388-400, 1993.

[98] B. Lerner, H. Guterman, M. Aladjem, and I. Dinstein, "A comparative study of neural network based feature extraction paradigms," *Pattern Recognition Letters*, vol. 20, no. 1, pp. 7-14, 1999.

[99] D.R. Lovell, C.R. Dance, M. Niranjan, R.W. Prager, K.J. Dalton, and R. Derom, "Feature selection using expected attainable discrimination," *Pattern Recognition Letters*, vol. 19, no. 5-6, pp. 393-402, 1998.

[100] D. Lowe and A. R. Webb, "Optimized feature extraction and the Bayes decision in feed-forward classifier networks," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 355-264, 1991.

[101] D.J.C. MacKay, "The evidence framework applied to classification networks," *Neural Computation*, vol. 4, no. 5, pp. 720-736, 1992.

[102] J. C. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Trans. on Neural Networks*, vol. 6, no. 2, pp. 296-317, 1995.

[103] J. Mao, K. Mohiuddin, and A. K. Jain, "Parsimonious network design and feature selection through node pruning," *Proc. 12th ICPR*, Jerusalem, pp. 622-624, Oct. 1994.

[104] J. C. Mao and K. M. Mohiuddin, "Improving OCR performance using character degradation models and boosting algorithm," *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1415-1419, 1997.

[105] G. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*, John Wiley & Sons, New York, 1992.

[106] M. Mehta, J. Rissanen, and R. Agrawal, "MDL-based decision tree pruning," *Proc. of the 1st Intl. Conf. on Knowledge Discovery in databases and Data Mining*, Montreal, Camada, Aug. 1995.

[107] C.E. Metz, "Basic Principles of ROC Analysis," *Seminars in Nuclear Medicine*, Vol. VIII, No. 4, pp. 283-298, 1978.

[108] R. S. Michalski and R. E. Stepp, "Automated construction of classifications: Conceptual clustering versus numerical taxonomy," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, pp. 396-410, 1983.

[109] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, New York, 1994.

[110] S. K. Mishra and V. V. Raghavan, "An empirical study of the performance of heuristic methods for clustering," in *Pattern Recognition in Practice*, E. S. Gelsema and L. N. Kanal, eds., Notrh-Holland, pp. 425-436, 1994.

[111] G. Nagy, "State of the art in pattern recognition," *Proc. IEEE*, vol. 56, pp. 836-862, 1968.

[112] G. Nagy, "Candide's practical principles of experimental pattern recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, no. 2, pp. 199-200, 1983.

[113] R. Neal, *Bayesian Learning for Neural Networks*, Spring Verlag, N. Y., 1996.

[114] H. Niemann, "Linear and nonlinear mappings of patterns," *Pattern Recognition*, vol. 12, pp. 83-87, 1980.

[115] K. L. Oehler and R. M. Gray, "Combining image compression and classification using vector quantization," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 5, pp. 461-473, 1995.

[116] E. Oja, *Subspace Methods of Pattern Recognition*, Research Studies Press, Letchworth, Hertfordshire, England, 1983.

[117] E. Oja, "Principal components, minor components, and linear neural networks," *Neural Networks*, vol. 5, no. 6, pp. 927-936, 1992.

[118] E. Oja, "The nonlinear PCA learning rule in independent component analysis," *Neurocomputing*, vol. 17, no. 1, pp. 25-45, 1997.

[119] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Proc. of IEEE Workshop on Neural Networks for Signal Processing*, J. Principe, L. Gile, N. Morgan, and E. Wilson, eds., IEEE Press, New York, 1997.

[120] Y. Park and J. Sklanski, "Automated design of linear tree classifiers," *Pattern Recognition*, vol. 23, no. 12, pp. 1393-1412, 1990.

[121] T. Pavlidis, *Structural Pattern Recognition*, Springer-Verlag, New York, 1977.

[122] L. I. Perlovsky, "Conundrum of combinatorial complexity," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 20, no. 6, pp. 666-670, 1998.

[123] M. P. Perrone and L. N. Cooper, "When networks disagree: Ensemble methods for hybrid neural networks," in *Neural Networks for Speech and Image Processing*, R. J. Mammone, ed., Chapman-Hall, 1993.

[124] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods – Support Vector Learning*, B. Scholkopf, C.J.C. Burges, and A.J. Smola, eds., MIT Press, Cambridge, MA, 1999.

[125] R. Picard, *Affective computing*, MIT Press, 1997.

[126] P. Pudil, J. Novovicova, and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119-1125, 1994.

[127] P. Pudil, J. Novovicova, and J. Kittler, "Feature selection based on the approximation of class densities by finite mixtures of the special type," *Pattern Recognition*, vol. 28, no. 9, pp. 1389-1398, 1995.

[128] J. R. Quinlan, "Simplifying decision trees," *Int. J. Man - Machine Studies*, vol. 27, pp. 221-234, 1987.

[129] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[130] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, pp. 257-286, Feb. 1989.

[131] S. J. Raudys and V. Pikelis, "On Dimensionality, Sample Size, Classification Error, and Complexity of Classification Algorithms in Pattern Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 2, pp. 243-251, 1980.

[132] S. J. Raudys and A. K. Jain, "Small sample size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252-264, 1991.

[133] S. Raudys, "Evolution and generalization of a single neuron; single-layer perceptron as seven statistical classifiers," *Neural Networks*, vol. 11, no. 2, pp. 283-296, 1998.

[134] S. Raudys and R. P. W. Duin, "Expected classification error of the Fisher linear classifier with pseudo-inverse covariance matrix," *Pattern Recognition Letters*, vol. 19, no. 5-6, pp. 385-392, 1998.

[135] S. Richardson and P. Green, "On Bayesian analysis of mixtures with unknown number of components," *Journal of the Royal Statistical Society (B)*, vol. 59, pp.731-792, 1997.

[136] B. Ripley, "Statistical aspects of neural networks," in *Networks on Chaos: Statistical and Probabilistic Aspects*, U. Bornndorff-Nielsen, J. Jensen, and W. Kendal, eds., Chapman and Hall, 1993.

[137] B. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge, 1996.

[138] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*, World Scientific Publishing Company, Singapore, 1989.

[139] K. Rose, "Deterministic annealing for clustering, compression, classification, regression and related optimization problems," *Proc. IEEE*, vol. 86, pp. 2210-2239, 1998.

[140] P. E. Ross, "Flash of Genius," *Forbes*, pp. 98-104, Nov. 16, 1998.

[141] J. W. Sammon Jr., "A nonlinear mapping for data structure analysis," *IEEE Trans. Comp.*, vol. C-l8, pp. 401-409, 1969.

[142] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, pp. 197-227, 1990.

[143] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," *Annals of Statistics*, 1999, in press.

[144] B. Schölkopf, *Support Vector Learning*, Ph.D. thesis, Technische Universität, Berlin, 1997.

[145] B. Schölkopf, A. Smola, and K. R. Muller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299-1319, 1998.

[146] B. Schölkopf, K.-K. Sung, C. J. C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with gaussian kernels to radial basis function classifiers," *IEEE Trans. on Signal Processing*, vol. 45, no. 11, pp. 2758-2765, 1997.

[147] J. Schurmann, *Pattern Classification: A Unified View of Statistical and Neural Approaches*, John Wiley & Sons, New York, 1996.

[148] S. Sclove, "Application of the conditional population mixture model to image segmentation," *IEEE Trans. Pattern Recognition and Machine Intelligence*, vol. 5, pp. 428-433, 1983.

[149] I. K. Sethi and G. P. R. Sarvarayudu, "Hierarchical classifier design using mutual information," *IEEE Trans. Pattern Recognition and Machine Intelligence*, vol. 1, pp. 194-201, April 1979.

[150] R. Setiono and H. Liu, "Neural-network feature selector," *IEEE Trans. of Neural Networks*, vol. 8, no. 3, pp. 654-662, 1997.

[151] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recognition Letters*, vol. 10, pp. 335-347, 1989.

[152] P. Simard, Y. LeCun, and J. Denker, "Efficient pattern recognition using a new transformation distance," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., Morgan Kaufmann, CA, 1993.

[153] P. Simard, B. Victorri, Y. LeCun, and J. Denker, "Tangent prop - a formalism for specifying selected invariances in an adaptive network," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, eds., pp. 651-655, Morgan Kaufmann, CA, 1992.

[154] P. Somol, P. Pudil, J. Novovicova, and P. Paclik, "Adaptive floating search methods in feature selection," *Proc. of PRIP-VI workshop*, Vlieland, June 1999.

[155] D. Titterington, A. Smith, and U. Makov, *Statistical Analysis of Finite Mixture Distributions*, John Wiley & Sons, Chichester (U.K.), 1985.

[156] V. Tresp and M. Taniguchi, "Combining estimators using non-constant weighting functions," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. S. Touretzky, and T. K. Leen, eds., vol. 7, MIT Press, Cambridge, MA, 1995.

[157] G. V. Trunk, "A problem of dimensionality: A simple example," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 1, no. 3, pp. 306-307, July 1979.

[158] K. Tumer and J. Ghosh, "Analysis of decision boundaries in linearly combined neural classifiers," *Pattern Recognition*, vol. 29, pp. 341-348, 1996.

[159] S. Vaithyanathan and B. Dom, "Model selection in unsupervised learning with applications to document clustering." *Proc. of the 6th Intl. Conf. on Machine Learning*, pp. 433-443, Bled, Slovenia, June 1999.

[160] M. van Breukelen, R. P. W. Duin, D. M. J. Tax, and J. E. den Hartog, "Handwritten digit recognition by combined classifiers," *Kybernetika*, vol. 34, no. 4, pp. 381-386, 1998.

[161] V. N. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer-Verlag, Berlin, 1998.

[162] V. N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, New York, 1998.

[163] S. Watanabe, *Pattern Recognition: Human and Mechanical*, Wiley, New York, 1985.

[164] S. Watanabe (ed.), *Frontiers of Pattern Recognition*, Academic Press, NY, 1972.

[165] A. R. Webb, "Multidimensional scaling by iterative majorization using radial basis functions," *Pattern Recognition*, vol. 28, no. 5, pp. 753-759, 1995.

[166] S. M. Weiss and C. A. Kulikowski, *Computer Systems that Learn*, Morgan Kaufmann Publishers, California, 1991.

[167] M. Whindham and A. Cutler, "Information ratios for validating mixture analysis," *J. Amer. Statist. Ass.*, vol. 87, pp. 1188-1192, 1992.

[168] D. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241-259, 1992.

[169] A. K. C. Wong and D. C. C. Wang, "DECA: a discrete-valued data clustering algorithm," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 1, pp. 342-349, 1979.

[170] K. Woods, W. P. Kegelmeyer Jr., and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 405-410, April 1997.

[171] Q. B. Xie, C. A. Laszlo, and R. K. Ward, "Vector quantization technique for nonparametric classifier design," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 15, no. 12, pp. 1326-1330, 1993.

[172] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods for combining multiple classifiers and their applications in handwritten character recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 418-435, 1992.

List of Figures

List of Tables