Combining Pattern Classifiers

Combining Pattern Classifiers

Methods and Algorithms

Ludmila I. Kuncheva

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Kuncheva, Ludmila I. (Ludmila Ilieva), 1959–
Combining pattern classifiers: methods and algorithms / Ludmila I. Kuncheva.
p. cm.
"A Wiley-Interscience publication."
Includes bibliographical references and index.
ISBN 0-471-21078-1 (cloth)
1. Pattern recognition systems. 2. Image processing–Digital techniques. I. Title.

TK7882.P3K83 2004
006.4–dc22

2003056883

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents

Preface

Everyday life throws at us an endless number of pattern recognition problems: smells, images, voices, faces, situations, and so on. Most of these problems we solve at a sensory level or intuitively, without an explicit method or algorithm. As soon as we are able to provide an algorithm the problem becomes trivial and we happily delegate it to the computer. Indeed, machines have confidently replaced humans in many formerly difficult or impossible, now just tedious pattern recognition tasks such as mail sorting, medical test reading, military target recognition, signature verification, meteorological forecasting, DNA matching, fingerprint recognition, and so on.

In the past, pattern recognition focused on designing single classifiers. This book is about combining the "opinions" of an ensemble of pattern classifiers in the hope that the new opinion will be better than the individual ones. "Vox populi, vox Dei."

The field of combining classifiers is like a teenager: full of energy, enthusiasm, spontaneity, and confusion; undergoing quick changes and obstructing the attempts to bring some order to its cluttered box of accessories. When I started writing this book, the field was small and tidy, but it has grown so rapidly that I am faced with the Herculean task of cutting out a (hopefully) useful piece of this rich, dynamic, and loosely structured discipline. This will explain why some methods and algorithms are only sketched, mentioned, or even left out and why there is a chapter called "Miscellanea" containing a collection of important topics that I could not fit anywhere else.

The book is not intended as a comprehensive survey of the state of the art of the whole field of combining classifiers. Its purpose is less ambitious and more practical: to expose and illustrate some of the important methods and algorithms. The majority of these methods are well known within the pattern recognition and machine

learning communities, albeit scattered across various literature sources and disguised under different names and notations. Yet some of the methods and algorithms in the book are less well known. My choice was guided by how intuitive, simple, and effective the methods are. I have tried to give sufficient detail so that the methods can be reproduced from the text. For some of them, simple Matlab code is given as well. The code is not foolproof nor is it optimized for time or other efficiency criteria. Its sole purpose is to enable the reader to experiment. Matlab was seen as a suitable language for such illustrations because it often looks like executable pseudocode.

I have refrained from making strong recommendations about the methods and algorithms. The computational examples given in the book, with real or artificial data, should not be regarded as a guide for preferring one method to another. The examples are meant to illustrate *how* the methods work. Making an extensive experimental comparison is beyond the scope of this book. Besides, the fairness of such a comparison rests on the conscientiousness of the designer of the experiment. J.A. Anderson says it beautifully [89]

> There appears to be imbalance in the amount of polish allowed for the techniques. There is a world of difference between "a poor thing – but my own" and "a poor thing but *his*".

The book is organized as follows. Chapter 1 gives a didactic introduction into the main concepts in pattern recognition, Bayes decision theory, and experimental comparison of classifiers. Chapter 2 contains methods and algorithms for designing the individual classifiers, called the base classifiers, to be used later as an ensemble. Chapter 3 discusses some philosophical questions in combining classifiers including: "Why should we combine classifiers?" and "How do we train the ensemble?" Being a quickly growing area, combining classifiers is difficult to put into unified terminology, taxonomy, or a set of notations. New methods appear that have to be accommodated within the structure. This makes it look like patchwork rather than a tidy hierarchy. Chapters 4 and 5 summarize the classifier fusion methods when the individual classifiers give label outputs or continuous-value outputs. Chapter 6 is a brief summary of a different approach to combining classifiers termed classifier selection. The two most successful methods for building classifier ensembles, *bagging* and *boosting*, are explained in Chapter 7. A compilation of topics is presented in Chapter 8. We talk about feature selection for the ensemble, error-correcting output codes (ECOC), and clustering ensembles. Theoretical results found in the literature are presented in Chapter 9. Although the chapter lacks coherence, it was considered appropriate to put together a list of such results along with the details of their derivation. The need of a general theory that underpins classifier combination has been acknowledged regularly, but such a theory does not exist as yet. The collection of results in Chapter 9 can be regarded as a set of jigsaw pieces awaiting further work. Diversity in classifier combination is a controversial issue. It is a necessary component of a classifier ensemble and yet its role in the ensemble performance is ambiguous. Little has been achieved by measuring diversity and

employing it for building the ensemble. In Chapter 10 we review the studies in diversity and join in the debate about its merit.

The book is suitable for postgraduate students and researchers in mathematics, statistics, computer science, engineering, operations research, and other related disciplines. Some knowledge of the areas of pattern recognition and machine learning will be beneficial.

The quest for a structure and a self-identity of the classifier combination field will continue. Take a look at any book for teaching introductory statistics; there is hardly much difference in the structure and the ordering of the chapters, even the sections and subsections. Compared to this, the classifier combination area is pretty chaotic. Curiously enough, questions like "Who needs a taxonomy?!" were raised at the Discussion of the last edition of the International Workshop on Multiple Classifier Systems, MCS 2003. I believe that we do need an explicit and systematic description of the field. How otherwise are we going to teach the newcomers, place our own achievements in the bigger framework, or keep track of the new developments? This book is an attempt to tidy up a piece of the classifier combination realm, maybe just the attic. I hope that, among the antiques, you will find new tools and, more importantly, new applications for the old tools.

LUDMILA I. KUNCHEVA

Bangor, Gwynedd, United Kingdom
September 2003

Acknowledgments

Many thanks to my colleagues from the School of Informatics, University of Wales, Bangor, for their support throughout this project. Special thanks to my friend Chris Whitaker with whom I have shared publications, bottles of wine, and many inspiring and entertaining discussions on multiple classifiers, statistics, and life. I am indebted to Bob Duin and Franco Masulli for their visits to Bangor and for sharing with me their time, expertise, and exciting ideas about multiple classifier systems. A great part of the material in this book was inspired by the Workshops on Multiple Classifier Systems held annually since 2000. Many thanks to Fabio Roli, Josef Kittler, and Terry Windeatt for keeping these workshops alive and fun. I am truly grateful to my husband Roumen and my daughters Diana and Kamelia for their love, patience, and understanding.

L. I. K.

Notation and Acronyms

CART	classification and regression trees
LDC	linear discriminant classifier
MCS	multiple classifier systems
PCA	principal component analysis
pdf	probability density function
pmf	probability mass function
QDC	quadratic discriminant classifier
RDA	regularized discriminant analysis
SVD	singular value decomposition
$P(A)$	a general notation for the probability of an event A
$P(A\|B)$	a general notation for the probability of an event A conditioned by an event B
$\mathcal{D}$	a classifier ensemble, $\mathcal{D} = \{D_1, \ldots, D_L\}$
D_i	an individual classifier from the ensemble
$\mathcal{E}$	the expectation operator
$\mathcal{F}$	an aggregation method or formula for combining classifier outputs
$\mathcal{I}(a, b)$	indicator function taking value 1 if $a = b$, and 0, otherwise
$\mathcal{I}(l(\mathbf{z}_j),\omega_i)$	example of using the indicator function: takes value 1 if the label of $\mathbf{z}_j$ is ω_i, and 0, otherwise
$l(\mathbf{z}_j)$	the class label of $\mathbf{z}_j$
$P(\omega_k)$	the prior probability for class ω_k to occur
$P(\omega_k\|\mathbf{x})$	the posterior probability that the true class is ω_k, given $\mathbf{x} \in \Re^n$

$p(\mathbf{x}\|\omega_k)$	the class-conditional probability density function for $\mathbf{x}$, given class ω_k
$\Re^n$	the n-dimensional real space
$\mathbf{s}$	the vector of the class labels produced by the ensemble, $\mathbf{s} = [s_1, \ldots, s_L]^T$
s_i	the class label produced by classifier D_i for a given input $\mathbf{x}$, $s_i \in \Omega$
$\mathcal{V}$	the variance operator
$\mathbf{x}$	a feature vector, $\mathbf{x} = [x_1, \ldots, x_n]^T$, $\mathbf{x} \in \Re^n$ (column vectors are always assumed)
$\mathbf{Z}$	the data set, $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, $\mathbf{z}_j \in \Re^n$, usually with known labels for all $\mathbf{z}_j$
Ω	the set of class labels, $\Omega = \{\omega_1, \ldots, \omega_c\}$
ω_k	a class label

1

Fundamentals of Pattern Recognition

1.1 BASIC CONCEPTS: CLASS, FEATURE, AND DATA SET

1.1.1 Pattern Recognition Cycle

Pattern recognition is about assigning labels to objects. Objects are described by a set of measurements called also attributes or features. Current research builds upon foundations laid out in the 1960s and 1970s. A series of excellent books of that time shaped up the outlook of the field including [1–10]. Many of them were subsequently translated into Russian. Because pattern recognition is faced with the challenges of solving real-life problems, in spite of decades of productive research, elegant modern theories still coexist with ad hoc ideas, intuition and guessing. This is reflected in the variety of methods and techniques available to the researcher.

Figure 1.1 shows the basic tasks and stages of pattern recognition. Suppose that there is a hypothetical User who presents us with the problem and a set of data. Our task is to clarify the problem, translate it into pattern recognition terminology, solve it, and communicate the solution back to the User.

If the data set is not given, an experiment is planned and a data set is collected. The relevant features have to be nominated and measured. The feature set should be as large as possible, containing even features that may not seem too relevant at this stage. They might be relevant in combination with other features. The limitations for the data collection usually come from the financial side of the project. Another possible reason for such limitations could be that some features cannot be easily

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

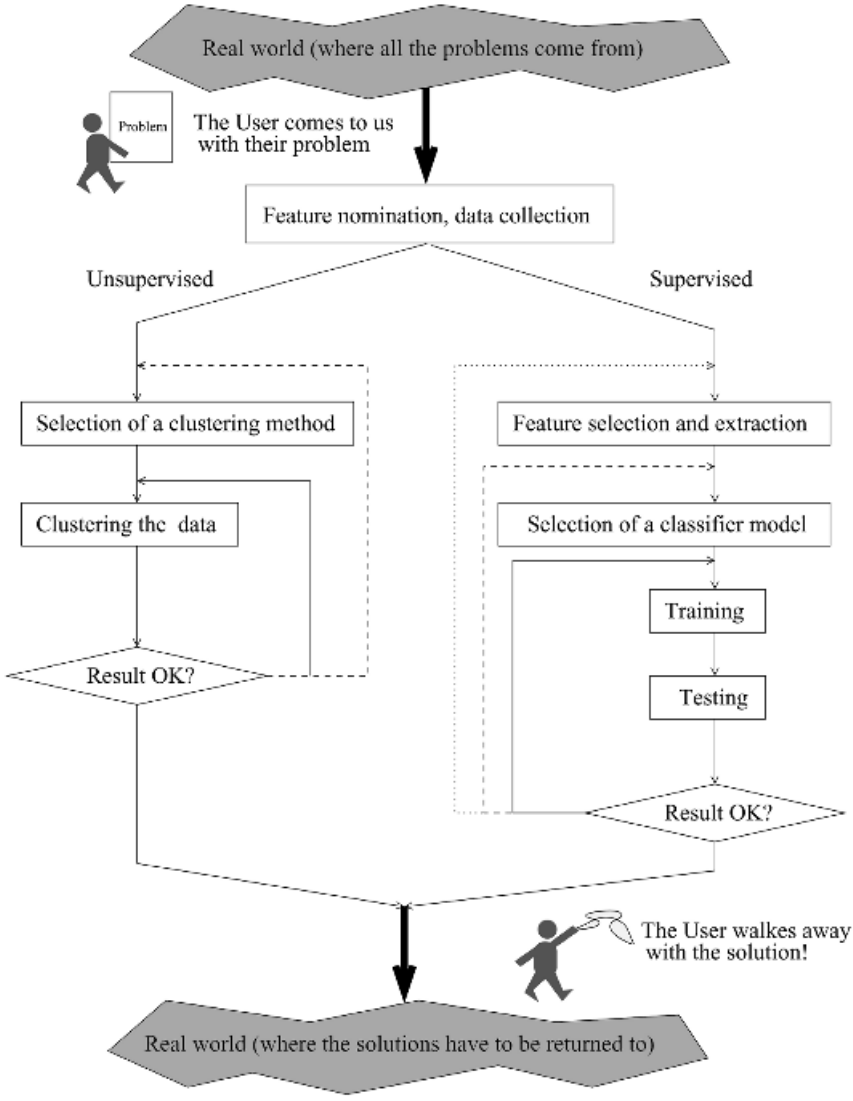


Fig. 1.1 *The pattern recognition cycle.*

measured, for example, features that require damaging or destroying the object, medical tests requiring an invasive examination when there are counterindications for it, and so on.

There are two major types of pattern recognition problems: unsupervised and supervised. In the unsupervised category (called also *unsupervised learning*), the problem is to discover the structure of the data set if there is any. This usually means that the User wants to know whether there are groups in the data, and what characteristics make the objects similar within the group and different across the

groups. Many clustering algorithms have been and are being developed for unsupervised learning. The choice of an algorithm is a matter of designer's preference. Different algorithms might come up with different structures for the same set of data. The curse and the blessing of this branch of pattern recognition is that there is no ground truth against which to compare the results. The only indication of how good the result is, is probably the subjective estimate of the User.

In the supervised category (called also *supervised learning*), each object in the data set comes with a preassigned class label. Our task is to train a *classifier* to do the labeling "sensibly." Most often the labeling process cannot be described in an algorithmic form. So we supply the machine with learning skills and present the labeled data to it. The classification knowledge learned by the machine in this process might be obscure, but the recognition accuracy of the classifier will be the judge of its adequacy.

Features are not all equally relevant. Some of them are important only in relation to others and some might be only "noise" in the particular context. Feature selection and extraction are used to improve the quality of the description.

Selection, training, and testing of a classifier model form the core of supervised pattern recognition. As the dashed and dotted lines in Figure 1.1 show, the loop of tuning the model can be closed at different places. We may decide to use the same classifier model and re-do the training only with different parameters, or to change the classifier model as well. Sometimes feature selection and extraction are also involved in the loop.

When a satisfactory solution has been reached, we can offer it to the User for further testing and application.

1.1.2 Classes and Class Labels

Intuitively, a class contains similar objects, whereas objects from different classes are dissimilar. Some classes have a clear-cut meaning, and in the simplest case are mutually exclusive. For example, in signature verification, the signature is either genuine or forged. The true class is one of the two, no matter that we might not be able to guess correctly from the observation of a particular signature. In other problems, classes might be difficult to define, for example, the classes of left-handed and right-handed people. Medical research generates a huge amount of difficulty in interpreting data because of the natural variability of the object of study. For example, it is often desirable to distinguish between low risk, medium risk, and high risk, but we can hardly define sharp discrimination criteria between these class labels.

We shall assume that there are c possible classes in the problem, labeled ω_1 to ω_c, organized as a set of labels $\Omega = \{\omega_1, \ldots, \omega_c\}$ and that each object belongs to one and only one class.

1.1.3 Features

As mentioned before, objects are described by characteristics called *features*. The features might be qualitative or quantitative as illustrated on the diagram in

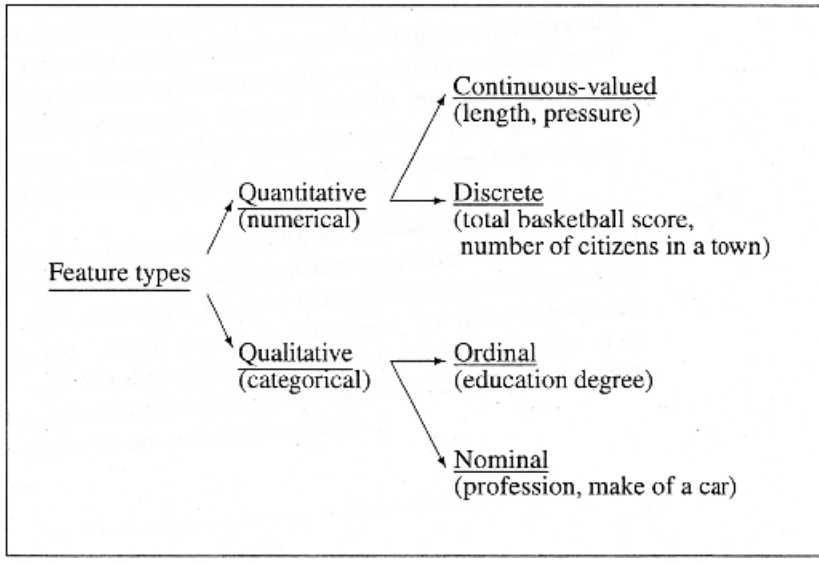


Fig. 1.2 *Types of features.*

Figure 1.2. Discrete features with a large number of possible values are treated as quantitative. Qualitative (categorical) features are these with small number of possible values, either with or without gradations. A branch of pattern recognition, called syntactic pattern recognition (as opposed to statistical pattern recognition) deals exclusively with qualitative features [3].

Statistical pattern recognition operates with numerical features. These include, for example, systolic blood pressure, speed of the wind, company's net profit in the past 12 months, gray-level intensity of a pixel. The feature values for a given object are arranged as an n-dimensional vector $\mathbf{x} = [x_1, \ldots, x_n]^T \in \Re^n$. The real space $\Re^n$ is called the *feature space*, each axis corresponding to a physical feature. Real-number representation ($\mathbf{x} \in \Re^n$) requires a methodology to convert qualitative features into quantitative. Typically, such methodologies are highly subjective and heuristic. For example, sitting an exam is a methodology to quantify students' learning progress. There are also unmeasurable features that we, humans, can assess intuitively but hardly explain. These include sense of humor, intelligence, and beauty. For the purposes of this book, we shall assume that all features have numerical expressions.

Sometimes an object can be represented by multiple subsets of features. For example, in identity verification, three different sensing modalities can be used [11]: frontal face, face profile, and voice. Specific feature subsets are measured for each modality and then the feature vector is composed by three subvectors, $\mathbf{x} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}]^T$. We call this *distinct pattern representation* after Kittler et al. [11]. As we shall see later, an ensemble of classifiers can be built using distinct pattern representation, one classifier on each feature subset.

Fig. 1.3 *Examples of handwritten digits.*

1.1.4 Data Set

The information to design a classifier is usually in the form of a labeled *data set* $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, $\mathbf{z}_j \in \Re^n$. The class label of $\mathbf{z}_j$ is denoted by $l(\mathbf{z}_j) \in \Omega$, $j = 1, \ldots, N$. Figure 1.3 shows a set of examples of handwritten digits, which have to be labeled by the machine into 10 classes. To construct a data set, the black and white images have to be transformed into feature vectors. It is not always easy to formulate the n features to be used in the problem. In the example in Figure 1.3, various discriminatory characteristics can be nominated, using also various transformations on the image. Two possible features are, for example, the number of vertical strokes and the number of circles in the image of the digit. Nominating a good set of features predetermines to a great extent the success of a pattern recognition system. In this book we assume that the features have been already defined and measured and we have a ready-to-use data set $\mathbf{Z}$.

1.2 CLASSIFIER, DISCRIMINANT FUNCTIONS, AND CLASSIFICATION REGIONS

A *classifier* is any function:

$$D : \Re^n \to \Omega \tag{1.1}$$

In the "canonical model of a classifier" [2] shown in Figure 1.4, we consider a set of c *discriminant functions* $G = \{g_1(\mathbf{x}), \ldots, g_c(\mathbf{x})\}$,

$$g_i : \Re^n \to \Re, \qquad i = 1, \ldots, c \tag{1.2}$$

each yielding a score for the respective class. Typically (and most naturally), $\mathbf{x}$ is labeled in the class with the highest score. This labeling choice is called the

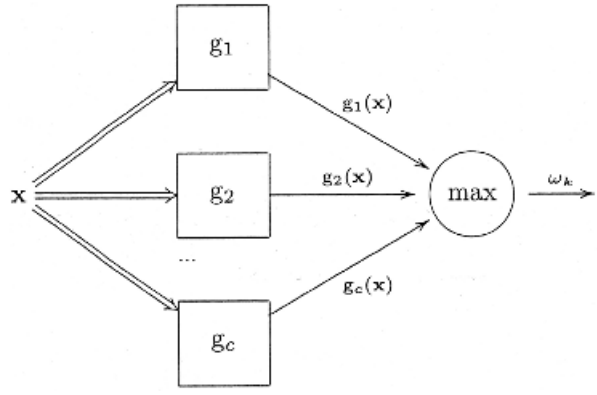


Fig. 1.4 *Canonical model of a classifier. The double arrows denote the n-dimensional input vector* $\mathbf{x}$*, the output of the boxes are the discriminant function values,* $g_i(x)$ *(scalars), and the output of the maximum selector is the class label* $\omega_k \in \Omega$ *assigned according to the maximum membership rule.*

maximum membership rule, that is,

$$D(\mathbf{x}) = \omega_{i^*} \in \Omega \Leftrightarrow g_{i^*}(\mathbf{x}) = \max_{i=1,\ldots,c} \{g_i(\mathbf{x})\} \tag{1.3}$$

Ties are broken randomly, that is, $\mathbf{x}$ is assigned randomly to one of the tied classes.

The discriminant functions partition the feature space $\Re^n$ into c (not necessarily compact) *decision regions* or *classification regions* denoted by $\mathcal{R}_1, \ldots, \mathcal{R}_c$

$$\mathcal{R}_i = \left\{ \mathbf{x} \,\middle|\, \mathbf{x} \in \Re^n, \, g_i(\mathbf{x}) = \max_{k=1,\ldots,c} g_k(\mathbf{x}) \right\}, \qquad i = 1, \ldots, c \tag{1.4}$$

The decision region for class ω_i is the set of points for which the ith discriminant function has the highest score. According to the maximum membership rule (1.3), all points in decision region $\mathcal{R}_i$ are assigned in class ω_i. The decision regions are specified by the classifier D, or, equivalently, by the discriminant functions G. The boundaries of the decision regions are called *classification boundaries*, and contain the points for which the highest discriminant function votes tie. A point on the boundary can be assigned to any of the bordering classes. If a decision region $\mathcal{R}_i$ contains data points from the labeled set $\mathbf{Z}$ with true class label $\omega_j, j \neq i$, the classes ω_i and ω_j are called *overlapping*. Note that overlapping classes for a particular partition of the feature space (defined by a certain classifier D) can be nonoverlapping if the feature space was partitioned in another way. If in $\mathbf{Z}$ there are no identical points with different class labels, we can *always* partition the feature space into

classification regions so that the classes are nonoverlapping. Generally, the smaller the overlapping, the better the classifier.

Example: Classification Regions. A 15-point two-class problem is depicted in Figure 1.5. The feature space $\Re^2$ is divided into two classification regions: $\mathcal{R}_1$ is shaded (class ω_1: squares) and $\mathcal{R}_2$ is not shaded (class ω_2: dots). For two classes we can use only one discriminant function instead of two:

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) \tag{1.5}$$

and assign class ω_1 if $g(\mathbf{x})$ is positive and class ω_2 if it is negative. For this example, we have drawn the classification boundary produced by the linear discriminant function

$$g(\mathbf{x}) = -7x_1 + 4x_2 + 21 = 0 \tag{1.6}$$

Notice that *any* line in $\Re^2$ is a linear discriminant function for any two-class problem in $\Re^2$. Generally, *any* set of functions $g_1(\mathbf{x}), \ldots, g_c(\mathbf{x})$ (linear or nonlinear) is a set of *discriminant functions*. It is another matter how successfully these discriminant functions separate the classes.

Let $G^* = \{g_1^*(\mathbf{x}), \ldots, g_c^*(\mathbf{x})\}$ be a set of *optimal* (in some sense) discriminant functions. We can obtain infinitely many sets of optimal discriminant functions from G^* by applying a transformation $f(g_i^*(\mathbf{x}))$ that preserves the order of the function values for every $\mathbf{x} \in \Re^n$. For example, $f(\zeta)$ can be a $\log(\zeta)$, $\sqrt{\zeta}$ for positive definite $g^*(\mathbf{x})$, a^{ζ}, for $a > 1$, and so on. Applying the same f to all discriminant functions in G^*, we obtain an equivalent set of discriminant functions. Using the

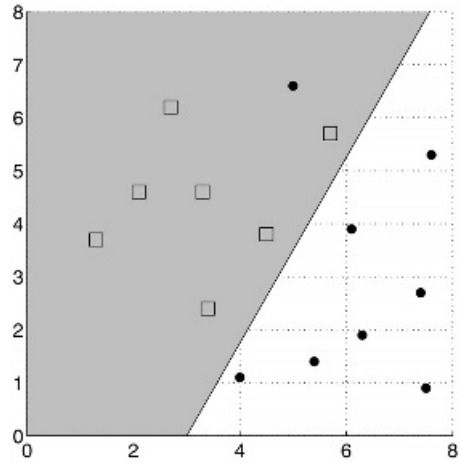


Fig. 1.5 *A two-class example with a linear discriminant function.*

maximum membership rule (1.3), $\mathbf{x}$ will be labeled to the same class by any of the equivalent sets of discriminant functions.

If the classes in $\mathbf{Z}$ can be separated completely from each other by a hyperplane (a point in $\Re$, a line in $\Re^2$, a plane in $\Re^3$), they are called *linearly separable*. The two classes in Figure 1.5 are not linearly separable because of the dot at (5,6.6) which is on the wrong side of the discriminant function.

1.3 CLASSIFICATION ERROR AND CLASSIFICATION ACCURACY

It is important to know how well our classifier performs. The *performance* of a classifier is a compound characteristic, whose most important component is the classification accuracy. If we were able to try the classifier on all possible input objects, we would know exactly how accurate it is. Unfortunately, this is hardly a possible scenario, so an estimate of the accuracy has to be used instead.

1.3.1 Calculation of the Error

Assume that a labeled data set $\mathbf{Z}_{ts}$ of size $N_{ts} \times n$ is available for testing the accuracy of our classifier, D. The most natural way to calculate an estimate of the error is to run D on all the objects in $\mathbf{Z}_{ts}$ and find the proportion of misclassified objects

$$\text{Error}(D) = \frac{N_{\text{error}}}{N_{ts}} \tag{1.7}$$

where N_{error} is the number of misclassifications committed by D. This is called the *counting estimator* of the error rate because it is based on the count of misclassifications. Let $s_j \in \Omega$ be the class label assigned by D to object $\mathbf{z}_j$. The counting estimator can be rewritten as

$$\text{Error}(D) = \frac{1}{N_{ts}} \sum_{j=1}^{N_{ts}} \big\{1 - \mathcal{I}(l(\mathbf{z}_j), s_j)\big\}, \qquad \mathbf{z}_j \in \mathbf{Z}_{ts} \tag{1.8}$$

where $\mathcal{I}(a, b)$ is an indicator function taking value 1 if $a = b$ and 0 if $a \neq b$. Error(D) is also called the *apparent error rate*. Dual to this characteristic is the apparent classification accuracy which is calculated by $1 - \text{Error}(D)$.

To look at the error from a probabilistic point of view, we can adopt the following model. The classifier commits an error with probability P_D on any object $\mathbf{x} \in \Re^n$ (a wrong but useful assumption). Then the number of errors has a binomial distribution with parameters (P_D, N_{ts}). An estimate of P_D is

$$\hat{P}_D = \frac{N_{\text{error}}}{N_{ts}} \tag{1.9}$$

which is in fact the counting error, Error(D), defined above. If N_{ts} and P_D satisfy the rule of thumb: $N_{ts} > 30$, $\hat{P}_D \times N_{ts} > 5$ and $(1 - \hat{P}_D) \times N_{ts} > 5$, the binomial distribution can be approximated by a normal distribution. The 95 percent confidence interval for the error is

$$\left[\hat{P}_D - 1.96\sqrt{\frac{\hat{P}_D(1-\hat{P}_D)}{N_{ts}}}, \quad \hat{P}_D + 1.96\sqrt{\frac{\hat{P}_D(1-\hat{P}_D)}{N_{ts}}} \right] \tag{1.10}$$

There are so-called *smoothing modifications* of the counting estimator [12] whose purpose is to reduce the variance of the estimate of P_D. The binary indicator function $\mathcal{I}(a, b)$ in Eq. (1.8) is replaced by a smoothing function taking values in the interval $[0, 1] \subset \Re$.

1.3.2 Training and Testing Data Sets

Suppose that we have a data set $\mathbf{Z}$ of size $N \times n$, containing n-dimensional feature vectors describing N objects. We would like to use as much as possible of the data to build the classifier (*training*), and also as much as possible unseen data to test its performance more thoroughly (*testing*). However, if we use all data for training and the same data for testing, we might *overtrain* the classifier so that it perfectly learns the available data and fails on unseen data. That is why it is important to have a separate data set on which to examine the final product. The main alternatives for making the best use of $\mathbf{Z}$ can be summarized as follows.

- *Resubstitution* (R-method). Design classifier D on $\mathbf{Z}$ and test it on $\mathbf{Z}$. $\hat{P}_D$ is optimistically biased.
- *Hold-out* (H-method). Traditionally, split $\mathbf{Z}$ into halves, use one half for training, and the other half for calculating $\hat{P}_D$. $\hat{P}_D$ is pessimistically biased. Splits in other proportions are also used. We can swap the two subsets, get another estimate $\hat{P}_D$ and average the two. A version of this method is the *data shuffle* where we do L random splits of $\mathbf{Z}$ into training and testing parts and average all L estimates of P_D calculated on the respective testing parts.
- *Cross-validation* (called also the *rotation method* or *π-method*). We choose an integer K (preferably a factor of N) and randomly divide $\mathbf{Z}$ into K subsets of size N/K. Then we use one subset to test the performance of D trained on the union of the remaining $K - 1$ subsets. This procedure is repeated K times, choosing a different part for testing each time. To get the final value of $\hat{P}_D$ we average the K estimates. When $K = N$, the method is called the *leave-one-out* (or *U-method*).
- *Bootstrap*. This method is designed to correct the optimistic bias of the R-method. This is done by randomly generating L sets of cardinality N from the original set $\mathbf{Z}$, with replacement. Then we assess and average the error rate of the classifiers built on these sets.

The question about the best way to organize the training/testing experiment has been around for a long time [13]. Pattern recognition has now outgrown the stage where the computation resource was the decisive factor as to which method to use. However, even with modern computing technology, the problem has not disappeared. The ever growing sizes of the data sets collected in different fields of science and practice pose a new challenge. We are back to using the good old hold-out method, first because the others might be too time-consuming, and secondly, because the amount of data might be so excessive that small parts of it will suffice for training and testing. For example, consider a data set obtained from retail analysis, which involves hundreds of thousands of transactions. Using an estimate of the error over, say, 10,000 data points, can conveniently shrink the confidence interval, and make the estimate reliable enough.

It is now becoming a common practice to use three instead of two data sets: one for training, one for *validation*, and one for testing. As before, the testing set remains unseen during the training process. The validation data set acts as pseudo-testing. We continue the training process until the performance improvement on the training set is no longer matched by a performance improvement on the validation set. At this point the training should be stopped so as to avoid overtraining. Not all data sets are large enough to allow for a validation part to be cut out. Many of the data sets from the UCI Machine Learning Repository Database (at *http://www.ics.uci.edu/~mlearn/MLRepository.html*), often used as benchmarks in pattern recognition and machine learning, may be unsuitable for a three-way split into training/validation/testing. The reason is that the data subsets will be too small and the estimates of the error on these subsets would be unreliable. Then stopping the training at the point suggested by the validation set might be inadequate, the estimate of the testing accuracy might be inaccurate, and the classifier might be poor because of the insufficient training data.

When multiple training and testing sessions are carried out, there is the question about which of the classifiers built during this process we should use in the end. For example, in a 10-fold cross-validation, we build 10 different classifiers using different data subsets. The above methods are only meant to give us an estimate of the accuracy of a certain model built for the problem at hand. We rely on the assumption that the classification accuracy will change smoothly with the changes in the size of the training data [14]. Therefore, if we are happy with the accuracy and its variability across different training subsets, we may decide finally to train a single classifier on the whole data set. Alternatively, we may keep the classifiers built throughout the training and consider using them together in an ensemble, as we shall see later.

1.3.3 Confusion Matrices and Loss Matrices

To find out how the errors are distributed across the classes we construct a *confusion matrix* using the testing data set, $\mathbf{Z}_{ts}$. The entry a_{ij} of such a matrix denotes the number of elements from $\mathbf{Z}_{ts}$ whose true class is ω_i, and which are assigned by D to class ω_j.

The confusion matrix for the linear classifier for the 15-point data depicted in Figure 1.5, is given as

	$D(\mathbf{x})$	
True Class	ω_1	ω_2
ω_1	7	0
ω_2	1	7

The estimate of the classifier's accuracy can be calculated as the trace of the matrix divided by the total sum of the entries, $(7 + 7)/15$ in this case. The additional information that the confusion matrix provides is where the misclassifications have occurred. This is important for problems with a large number of classes because a large off-diagonal entry of the matrix might indicate a difficult two-class problem that needs to be tackled separately.

Example: Confusion Matrix for the Letter Data. The Letter data set available from the UCI Machine Learning Repository Database contains data extracted from 20,000 black and white images of capital English letters. Sixteen numerical features describe each image ($N = 20{,}000$, $c = 26$, $n = 16$). For the purpose of this illustration we used the hold-out method. The data set was randomly split into halves. One half was used for training a linear classifier, and the other half was used for testing. The labels of the testing data were matched to the labels obtained from the classifier, and the 26×26 confusion matrix was constructed. If the classifier was ideal and all labels matched, the confusion matrix would be diagonal.

Table 1.1 shows the row in the confusion matrix corresponding to class "H." The entries show the number of times that true "H" is mistaken for the letter in the respective column. The boldface number is the diagonal entry showing how many times "H" has been correctly recognized. Thus, from the total of 379 examples of "H" in the testing set, only 165 have been labeled correctly by the classifier. Curiously, the largest number of mistakes, 37, are for the letter "O."

The errors in classification are not equally costly. To account for the different costs of mistakes, we introduce the *loss matrix*. We define a loss matrix with entries

TABLE 1.1 The "H"-Row in the Confusion Matrix for the Letter Data Set Obtained from a Linear Classifier Trained on 10,000 Points.

"H" mistaken for:	A	B	C	D	E	F	G	H	I	J	K	L	M
No of times:	2	12	0	27	0	2	1	**165**	0	0	26	0	1
"H" mistaken for:	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
No of times:	31	37	4	8	17	1	1	13	3	1	27	0	0

λ_{ij} denoting the loss incurred by assigning label ω_i, given that the true label of the object is ω_j. If the classifier is unsure about the label, it may refuse to make a decision. An extra class (called refuse-to-decide) denoted ω_{c+1} can be added to Ω. Choosing ω_{c+1} should be less costly than choosing a wrong class. For a problem with c original classes and a refuse option, the loss matrix is of size $(c+1) \times c$. Loss matrices are usually specified by the user. A zero–one (0–1) loss matrix is defined as $\lambda_{ij} = 0$ for $i = j$ and $\lambda_{ij} = 1$ for $i \neq j$, that is, all errors are equally costly.

1.4 EXPERIMENTAL COMPARISON OF CLASSIFIERS

There is no single "best" classifier. Classifiers applied to different problems and trained by different algorithms perform differently [15–17]. Comparative studies are usually based on extensive experiments using a number of simulated and real data sets. Dietterich [14] details four important sources of variation that have to be taken into account when comparing classifier models.

1. *The choice of the testing set.* Different testing sets may rank differently classifiers that otherwise have the same accuracy across the whole population. Therefore it is dangerous to draw conclusions from a single testing experiment, especially when the data size is small.
2. *The choice of the training set.* Some classifier models are called *instable* [18] because small changes in the training set can cause substantial changes of the classifier trained on this set. Examples of instable classifiers are decision tree classifiers and some neural networks. (Note, all classifier models mentioned will be discussed later.) Instable classifiers are versatile models that are capable of adapting, so that all training examples are correctly classified. The instability of such classifiers is in a way the pay-off for their versatility. As we shall see later, instable classifiers play a major role in classifier ensembles. Here we note that the variability with respect to the training data has to be accounted for.
3. *The internal randomness of the training algorithm.* Some training algorithms have a random component. This might be the initialization of the parameters of the classifier, which are then fine-tuned (e.g., backpropagation algorithm for training neural networks), or a random-based procedure for tuning the classifier (e.g., a genetic algorithm). Thus the trained classifier might be different for the same training set and even for the same initialization of the parameters.
4. *The random classification error.* Dietterich [14] considers the possibility of having mislabeled objects in the testing data as the fourth source of variability.

The above list suggests that multiple training and testing sets should be used, and multiple training runs should be carried out for classifiers whose training has a stochastic element.

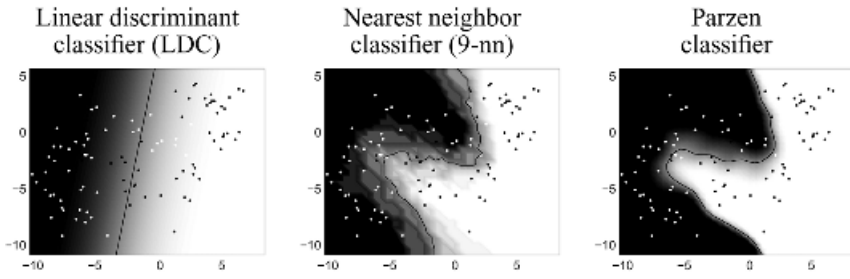


Fig. 1.6 *The decision regions found by the three classifiers.*

Let $\{D_1, \ldots, D_L\}$ be a set of classifiers tested on the same data set $\mathbf{Z}_{ts} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{N_{ts}}\}$. Let the classification results be organized in a binary $N_{ts} \times L$ matrix whose ijth entry is 0 if classifier D_j has misclassified vector $\mathbf{z}_i$, and 1 if D_j has produced the correct class label $l(\mathbf{z}_i)$.

Example: Arranging the Classifier Output Results. Figure 1.6 shows the classification regions of three classifiers trained to recognize two banana-shaped classes.[1] A training set was generated consisting of 100 points, 50 in each class, and another set of the same size was generated for testing. The points are uniformly distributed along the banana shapes with a superimposed normal distribution with a standard deviation 1.5.

The figure gives the gradient-shaded regions and a scatter plot of the testing data. The confusion matrices of the classifiers for the testing data are shown in Table 1.2. The classifier models and their training will be discussed further in the book. We are now only interested in comparing the accuracies.

The matrix with the correct/incorrect outputs is summarized in Table 1.3. The number of possible combinations of zeros and ones at the outputs of the three classifiers for a given object, for this example, is $2^L = 8$. The table shows the 0–1 combinations and the number of times they occur in $\mathbf{Z}_{ts}$.

The data in the table is then used to test statistical hypotheses about the equivalence of the accuracies.

1.4.1 McNemar and Difference of Proportion Tests

Suppose we have two trained classifiers that have been run on the same testing data giving testing accuracies of 98 and 96 percent, respectively. Can we claim that the first classifier is significantly better than the second?

[1] To generate the training and testing data sets we used the `gendatb` command from the PRTOOLS toolbox for Matlab [19]. This toolbox has been developed by Prof. R. P. W. Duin and his group (Pattern Recognition Group, Department of Applied Physics, Delft University of Technology) as a free aid for researchers in pattern recognition and machine learning. Available at *http://www.ph.tn.tudelft.nl/~bob/PRTOOLS.html*. Version 2 was used throughout this book.

TABLE 1.2 Confusion Matrices and Total Accuracies for the Three Classifiers on the Banana Data.

LDC		9-nm		Parzen	
42	8	44	6	47	3
8	42	2	48	5	45
84% correct		92% correct		92% correct	

LDC, linear discriminant classifier; 9-nn, nine nearest neighbor.

TABLE 1.3 Correct/Incorrect Outputs for the Three Classifiers on the Banana Data: "0" Means Misclassification, "1" Means Correct Classification.

D_1 = LDC	D_2 = 9-nn	D_3 = Parzen	Number
1	1	1	80
1	1	0	2
1	0	1	0
1	0	0	2
0	1	1	9
0	1	0	1
0	0	1	3
0	0	0	3
84	92	92	100

LDC, linear discriminant classifier; 9-nn, nine nearest neighbor.

1.4.1.1 McNemar Test. The testing results for two classifiers D_1 and D_2 are expressed in Table 1.4.

The null hypothesis, H_0, is that there is no difference between the accuracies of the two classifiers. If the null hypothesis is correct, then the expected counts for both off-diagonal entries in Table 1.4 are $\frac{1}{2}(N_{01} + N_{10})$. The discrepancy between the expected and the observed counts is measured by the following statistic

$$x^2 = \frac{(|N_{01} - N_{10}| - 1)^2}{N_{01} + N_{10}} \tag{1.11}$$

which is approximately distributed as χ^2 with 1 degree of freedom. The "-1" in the numerator is a continuity correction [14]. The simplest way to carry out the test is to calculate x^2 and compare it with the tabulated χ^2 value for, say, level of significance 0.05.[2]

[2] The *level of significance* of a statistical test is the probability of rejecting H_0 when it is true, that is, the probability to "convict the innocent." This error is called *Type I error*. The alternative error, when we do not reject H_0 when it is in fact incorrect, is called *Type II error*. The corresponding name for it would be "free the guilty." Both errors are needed in order to characterize a statistical test. For example, if we always accept H_0, there will be no Type I error at all. However, in this case the Type II error might be large. Ideally, both errors should be small.

TABLE 1.4 The 2 × 2 Relationship Table with Counts.

	D_2 correct (1)	D_2 wrong (0)
D_1 correct (1)	N_{11}	N_{10}
D_1 wrong (0)	N_{01}	N_{00}

Total, $N_{11} + N_{10} + N_{01} + N_{00} = N_{ts}$.

Then if $x^2 > 3.841$, we reject the null hypothesis and accept that the two classifiers have significantly different accuracies.

1.4.1.2 Difference of Two Proportions. Denote the two proportions of interest to be the accuracies of the two classifiers, estimated from Table 1.4 as

$$p_1 = \frac{N_{11} + N_{10}}{N_{ts}}; \qquad p_2 = \frac{N_{11} + N_{01}}{N_{ts}} \tag{1.12}$$

We can use Eq. (1.10) to calculate the 95 percent confidence intervals of the two accuracies, and if they are not overlapping, we can conclude that the accuracies are significantly different.

A shorter way would be to consider just one random variable, $d = p_1 - p_2$. We can approximate the two binomial distributions by normal distributions (given that $N_{ts} \geq 30$ and $p_1 \times N_{ts} > 5$, $(1 - p_1) \times N_{ts} > 5$, $p_2 \times N_{ts} > 5$, and $(1 - p_2) \times N_{ts} > 5$).

If the two errors are *independent*, then d is a normally distributed random variable. Under a null hypothesis, H_0, of equal p_1 and p_2, the following statistic

$$z = \frac{p_1 - p_2}{\sqrt{(2p(1-p))/(N_{ts})}} \tag{1.13}$$

has (approximately) a standard normal distribution, where $p = \frac{1}{2}(p_1 + p_2)$ is the pooled sample proportion. The null hypothesis is rejected if $|z| > 1.96$ (a two-sided test with a level of significance of 0.05).

Note that this test assumes that the testing experiments are done on independently drawn testing samples of size N_{ts}. In our case, the classifiers use the same testing data, so it is more appropriate to use a *paired* or *matched* test. Dietterich shows [14] that with a correction for this dependence, we arrive at a statistic that is the square root of x^2 in Eq. (1.11). Since the above z statistic is commonly used in the machine learning literature, Dietterich investigated experimentally how badly z is affected by the violation of the independence assumption. He recommends using the McNemar test rather than the difference of proportions test.

Example: Comparison of Two Classifiers on the Banana Data. Consider the linear discriminant classifier (LDC) and the nine-nearest neighbor classifier (9-nn) for the banana data. Using Table 1.3, we can construct the two-way table with

counts needed for the calculation of x^2 in Eq. (1.11).

$$N_{11} = 80 + 2 = 82 \qquad N_{10} = 0 + 2 = 2$$
$$N_{01} = 9 + 1 = 10 \qquad N_{00} = 3 + 3 = 6$$

From Eq. (1.11)

$$x^2 = \frac{(|10 - 2| - 1)^2}{10 + 2} = \frac{49}{12} \approx \mathbf{4.0833} \tag{1.14}$$

Since the calculated x^2 is greater than the tabulated value of 3.841, we reject the null hypothesis and accept that LDC and 9-nn are significantly different. Applying the difference of proportions test to the same pair of classifiers gives $p = (0.84 + 0.92)/2 = 0.88$, and

$$z = \frac{0.84 - 0.92}{\sqrt{(2 \times 0.88 \times 0.12)/(100)}} \approx \mathbf{-1.7408} \tag{1.15}$$

In this case $|z|$ is smaller than the tabulated value of 1.96, so we cannot reject the null hypothesis and claim that LDC and 9-nn have significantly different accuracies.

Which of the two decisions do we trust? The McNemar test takes into account the fact that the same testing set $\mathbf{Z}_{ts}$ was used whereas the difference of proportions does not. Therefore, we can accept the decision of the McNemar test. (Would it not have been better if the two tests agreed?)

1.4.2 Cochran's *Q* Test and *F*-Test

To compare $L > 2$ classifiers on the same testing data, the Cochran's Q test or the F-test can be used.

1.4.2.1 Cochran's Q Test. Cochran's Q test is proposed for measuring whether there are significant differences in L proportions measured on the same data [20]. This test is used in Ref. [21] in the context of comparing classifier accuracies. Let p_i denote the classification accuracy of classifier D_i. We shall test the hypothesis for no difference between the classification accuracies (equal proportions):

$$H_0 : p_1 = p_2 = \cdots = p_L \tag{1.16}$$

If there is no difference, then the following statistic is distributed approximately as χ^2 with $L - 1$ degrees of freedom

$$Q_C = (L - 1)\frac{L\sum_{i=1}^{L} G_i^2 - T^2}{LT - \sum_{j=1}^{N_{ts}} (L_j)^2} \tag{1.17}$$

where G_i is the number of objects out of N_{ts} correctly classified by D_i, $i = 1, \ldots, L$; L_j is the number of classifiers out of L that correctly classified object $\mathbf{z}_j \in \mathbf{Z}_{ts}$; and T is the total number of correct votes among the L classifiers

$$T = \sum_{i=1}^{L} G_i = \sum_{j=1}^{N_{ts}} L_j \tag{1.18}$$

To test H_0 we compare the calculated Q_C with the tabulated value of χ^2 for $L - 1$ degrees of freedom and the desired level of significance. If the calculated value is greater than the tabulated value, we reject the null hypothesis and accept that there are significant differences among the classifiers. We can apply pairwise tests to find out which pair (or pairs) of classifiers are significantly different.

1.4.2.2 F-Test. Looney [22] proposed a method for testing L independent classifiers on the same testing set. The sample estimates of the accuracies, $\bar{p}_1, \ldots, \bar{p}_L$, and $\bar{p}$ are found and used to calculate the sum of squares for the classifiers

$$SSA = N_{ts} \sum_{i=1}^{L} \bar{p}_i^2 - N_{ts} L \bar{p}^2 \tag{1.19}$$

and the sum of squares for the objects

$$SSB = \frac{1}{L} \sum_{j=1}^{N_{ts}} (L_j)^2 - L\, N_{ts} \bar{p}^2 \tag{1.20}$$

The total sum of squares is

$$SST = N_{ts} L \bar{p}(1 - \bar{p}) \tag{1.21}$$

and the sum of squares for classification–object interaction is

$$SSAB = SST - SSA - SSB \tag{1.22}$$

The *calculated* F value is obtained by

$$MSA = \frac{SSA}{(L-1)}; \qquad MSAB = \frac{SSAB}{(L-1)(N_{ts}-1)}; \qquad F_{\text{cal}} = \frac{MSA}{MSAB} \tag{1.23}$$

We check the validity of H_0 by comparing our F_{cal} with the tabulated value of an F-distribution with degrees of freedom $(L - 1)$ and $(L - 1) \times (N_{ts} - 1)$. If F_{cal} is greater than the tabulated F-value, we reject the null hypothesis H_0 and can further search for pairs of classifiers that differ significantly. Looney [22] suggests we use the same F_{cal} but with *adjusted* degrees of freedom (called the F^+ test).

Example: Cochran's Q Test and F-Test for Multiple Classifiers. For the three classifiers on the banana data, LDC, 9-nn, and Parzen, we use Table 1.3 to calculate $T = 84 + 92 + 92 = 268$, and subsequently

$$Q_C = 2 \times \frac{3 \times (84^2 + 92^2 + 92^2) - 268^2}{3 \times 268 - (80 \times 9 + 11 \times 4 + 6 \times 1)} \approx \mathbf{3.7647} \tag{1.24}$$

The tabulated value of χ^2 for $L - 1 = 2$ degrees of freedom and level of significance 0.05 is 5.991. Since the calculated value is smaller than that, we cannot reject H_0.

For the F-test, the results are

$$\begin{aligned}
SSA &= 100 \times (0.84^2 + 2 \times 0.92^2 - 3 \times 0.8933^2) \approx 0.4445 \\
SSB &= \tfrac{1}{3}(80 \times 9 + 11 \times 4 + 6 \times 1) - 3 \times 100 \times 0.8933^2 \approx 17.2712 \\
SST &= 100 \times 3 \times 0.8933 \times 0.1067 \approx 28.5945 \\
SSAB &= 28.5945 - 0.4445 - 17.2712 = 10.8788 \\
MSA &= 0.4445/2 \approx 0.2223 \\
MSAB &= 10.8788/(2 \times 99) \approx 0.0549 \\
F_{\text{cal}} &= \frac{0.2223}{0.0549} \approx \mathbf{4.0492}
\end{aligned}$$

The tabulated F-value for degrees of freedom 2 and $(2 \times 99) = 198$ is 3.09. In this example the F-test disagrees with the Cochran's Q test, suggesting that we can reject H_0 and accept that there is a significant difference among the three compared classifiers.

Looney [22] recommends the F-test because it is the less conservative of the two. Indeed, in our example, the F-test did suggest difference between the three classifiers whereas Cochran's Q test did not. Looking at the scatterplots and the classification regions, it seems more intuitive to agree with the tests that do indicate difference: the McNemar test (between LDC and 9-nn) and the F-test (among all three classifiers).

1.4.3 Cross-Validation Tests

We now consider several ways to account for the variability of the training and testing data.

1.4.3.1 K-Hold-Out Paired t-Test. According to Ref. [14], this test is widely used for comparing algorithms in machine learning. Consider two classifier models, A and B, and a data set $\mathbf{Z}$. The data set is split into training and testing subsets, usually $2/3$ for training and $1/3$ for testing (the hold-out method). Classifiers A and B are trained on the training set and tested on the testing set. Denote the

observed testing accuracies as P_A and P_B, respectively. This process is repeated K times (typical value of K is 30), and the testing accuracies are tagged with superscripts (i), $i = 1, \ldots, K$. Thus a set of K differences is obtained, $P^{(1)} = P_A^{(1)} - P_B^{(1)}$ to $P^{(K)} = P_A^{(K)} - P_B^{(K)}$. The assumption that we make is that the set of differences is an independently drawn sample from an approximately normal distribution. Then, under the null hypothesis (H_0: equal accuracies), the following statistic has a t-distribution with $K - 1$ degrees of freedom

$$t = \frac{\bar{P}\sqrt{K}}{\sqrt{\sum_{i=1}^{K} (P^{(i)} - \bar{P})^2/(K-1)}} \tag{1.25}$$

where $\bar{P} = (1/K)\sum_{i=1}^{K} P^{(i)}$. If the calculated t is greater than the tabulated value for the chosen level of significance and $K - 1$ degrees of freedom, we reject H_0 and accept that there are significant differences in the two compared classifier models. Dietterich argues that the above design might lead to deceptive results because the assumption of the *independently drawn* sample is invalid. The differences are dependent because the data sets used to train the classifier models and the sets to estimate the testing accuracies in each of the K runs are overlapping. This is found to be a severe drawback.

1.4.3.2 K-Fold Cross-Validation Paired t-Test. This is an alternative of the above procedure, which avoids the overlap of the testing data. The data set is split into K parts of approximately equal sizes, and each part is used in turn for testing of a classifier built on the pooled remaining $K - 1$ parts. The resultant differences are again assumed to be an independently drawn sample from an approximately normal distribution. The same statistic t, as in Eq. (1.25), is calculated and compared with the tabulated value.

Only part of the problem is resolved by this experimental set-up. The testing sets are independent, but the training sets are overlapping again. Besides, the testing set sizes might become too small, which entails high variance of the estimates.

1.4.3.3 Dietterich's 5 × 2-Fold Cross-Validation Paired t-Test (5 × 2cv). Dietterich [14] suggests a testing procedure that consists of repeating a two-fold cross-validation procedure five times. In each cross-validated run, we split the data into training and testing halves. Classifier models A and B are trained first on half #1, and tested on half #2, giving observed accuracies $P_A^{(1)}$ and $P_B^{(1)}$, respectively. By swapping the training and testing halves, estimates $P_A^{(2)}$ and $P_B^{(2)}$ are obtained. The differences are respectively

$$P^{(1)} = P_A^{(1)} - P_B^{(1)}$$

and

$$P^{(2)} = P_A^{(2)} - P_B^{(2)}$$

The estimated mean and variance of the differences, for this two-fold cross-validation run, are calculated as

$$\bar{P} = \frac{P^{(1)} + P^{(2)}}{2}; \qquad s^2 = \left(P^{(1)} - \bar{P}\right)^2 + \left(P^{(2)} - \bar{P}\right)^2 \tag{1.26}$$

Let $P_i^{(1)}$ denote the difference $P^{(1)}$ in the ith run, and s_i^2 denote the estimated variance for run i, $i = 1, \ldots, 5$. The proposed $\tilde{t}$ statistic is

$$\tilde{t} = \frac{P_1^{(1)}}{\sqrt{(1/5)\sum_{i=1}^{5} s_i^2}} \tag{1.27}$$

Note that only one of the ten differences that we will calculate throughout this experiment is used in the numerator of the formula. It is shown in Ref. [14] that under the null hypothesis, $\tilde{t}$ has approximately a t distribution with five degrees of freedom.

Example: Comparison of Two Classifier Models Through Cross-Validation Tests. The banana data set used in the previous examples is suitable for experimenting here because we can generate as many as necessary *independent* data sets from the same distribution. We chose the 9-nn and Parzen classifiers. The Matlab code for the three cross-validation methods discussed above is given in Appendices 1A to 1C at the end of this chapter. PRTOOLS toolbox for Matlab, version 2 [19], was used to train and test the two classifiers.

K-Hold-Out Paired t-Test. The training and testing data sets used in the previous example were pooled and the K-hold-out paired t-test was run with $K = 30$, as explained above. We chose to divide the data set into halves instead of a $2/3$ to $1/3$ split. The test statistic (1.25) was found to be $t = \mathbf{1.9796}$. At level of significance 0.05, and degrees of freedom $K - 1 = 29$, the tabulated value is 2.045 (two-tailed test). Since the calculated value is smaller than the tabulated value, we cannot reject the null hypothesis. This test suggests that 9-nn and Parzen classifiers do not differ in accuracy on the banana data. The averaged accuracies over the 30 runs were 92.5 percent for 9-nn, and 91.83 percent for Parzen.

K-Fold Cross-Validation Paired t-Test. We ran a 10-fold cross-validation for the set of 200 data points, so each testing set consisted of 20 objects. The ten testing accuracies for 9-nn and Parzen are shown in Table 1.5.

From Eq. (1.25) we found $t = \mathbf{1.0000}$. At level of significance 0.05, and degrees of freedom $K - 1 = 9$, the tabulated value is 2.262 (two-tailed test). Again, since the

TABLE 1.5 Accuracies (in %) of 9-nn and Parzen Using a 10-Fold Cross-Validation on the Banana Data.

	Sample #									
	1	2	3	4	5	6	7	8	9	10
9-nn (model *A*)	90	95	95	95	95	90	100	80	85	90
Parzen (model *B*)	90	95	95	95	95	90	100	85	85	90
$P_A - P_B$	0	0	0	0	0	0	0	−5	0	0

calculated value is smaller than the tabulated value, we cannot reject the null hypothesis, and we accept that 9-nn and Parzen do not differ in accuracy on the banana data. The averaged accuracies over the 10 splits were 91.50 percent for 9-nn, and 92.00 percent for Parzen.

5 × 2cv. The results of the five cross-validation runs are summarized in Table 1.6. Using (1.27), $\tilde{t} = \mathbf{1.0690}$. Comparing it with the tabulated value of 2.571 (level of significance 0.05, two-tailed test, five degrees of freedom), we again conclude that there is no difference in the accuracies of 9-nn and Parzen. The averaged accuracies across the 10 estimates (5 runs × 2 estimates in each) were 91.90 for 9-nn and 91.20 for Parzen.

Looking at the averaged accuracies in all three tests, it is tempting to conclude that 9-nn is marginally better than Parzen on this data. In many publications differences in accuracy are claimed on even smaller discrepancies. However, none of the three tests suggested that the difference is significant.

To re-confirm this result we ran a larger experiment where we did generate *independent* training and testing data sets from the same distribution, and applied the paired *t*-test as in Eq. (1.25). Now the assumptions of independence are satisfied and the test should be accurate. The Matlab code for this experiment is given in Appendix 1D at the end of this chapter. Five hundred training and testing samples, of size 100 each, were generated. The averaged accuracy over the 500 runs was 91.61 percent for 9-nn and 91.60 percent for the Parzen classifier. The *t*-statistic was calculated to be **0.1372** (we can use the standard normal distribution in this case because $K = 500 \geq 30$). The value is smaller than 1.96 (tabulated value for

TABLE 1.6 Accuracies (in %), Differences (in %), and Variances s^2 of 9-nn (*A*) and Parzen (*B*) Using a 5 × 2-Fold Cross-Validation on the Banana Data.

Exp #	$P_A^{(1)}$	$P_B^{(1)}$	$P^{(1)}$	$P_A^{(2)}$	$P_B^{(2)}$	$P^{(2)}$	s^2
1	93	91	2	93	94	−1	0.00045
2	92	89	3	93	93	0	0.00045
3	90	90	0	88	90	−2	0.00020
4	94	94	0	91	88	3	0.00045
5	93	93	0	92	90	2	0.00020

level of significance 0.05). Therefore we cannot conclude that there is a significant difference between the two models on this data set.

It is intuitively clear that simple models or stable classifiers are less likely to be overtrained than more sophisticated models. However, simple models might not be versatile enough to fit complex classification boundaries. More complex models (e.g., neural networks and prototype-based classifiers) have a better flexibility but require more system resources and are prone to overtraining. What do "simple" and "complex" mean in this context? The main aspects of complexity can be summarized as [23]

- training time and training complexity;
- memory requirements (e.g., the number of the parameters of the classifier that are needed for its operation); and
- running complexity.

1.4.4 Experiment Design

When talking about experiment design, I cannot refrain from quoting again and again a masterpiece of advice by George Nagy titled "Candide's practical principles of experimental pattern recognition" [24]:

> **Comparison of Classification Accuracies**
>
> Comparisons against algorithms proposed by others are distasteful and should be avoided. When this is not possible, the following Theorem of Ethical Data Selection may prove useful. Theorem: There exists a set of data for which a candidate algorithm is superior to any given rival algorithm. This set may be constructed by omitting from the test set any pattern which is misclassified by the candidate algorithm.
>
> **Replication of Experiments**
>
> Since pattern recognition is a mature discipline, the replication of experiments on new data by independent research groups, a fetish in the physical and biological sciences, is unnecessary. Concentrate instead on the accumulation of novel, universally applicable algorithms. Casey's Caution: Do not ever make your experimental data available to others; someone may find an obvious solution that you missed.

Albeit meant to be satirical, the above principles are surprisingly widespread and closely followed! Speaking seriously now, the rest of this section gives some practical tips and recommendations.

Example: Which Is the "Best" Result? Testing should be carried out on previously *unseen* data. Let $D(r)$ be a classifier with a parameter r such that varying r leads to different training accuracies. To account for this variability, here we use a randomly drawn 1000 objects from the Letter data set. The remaining

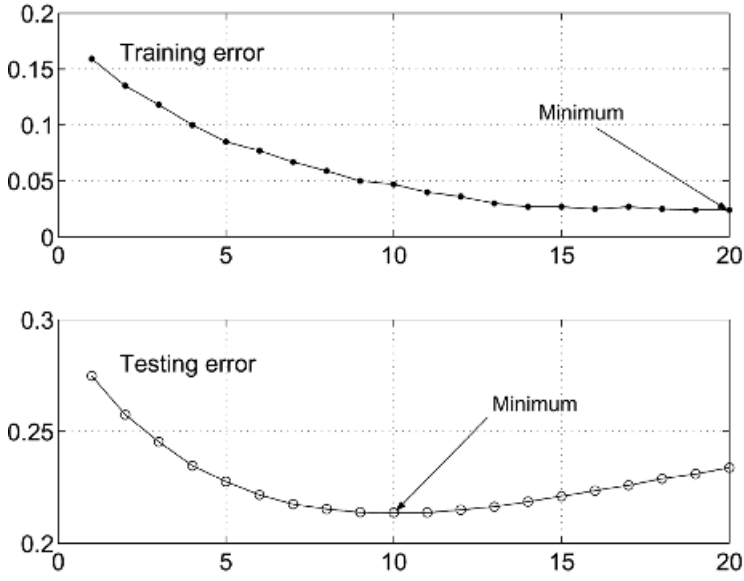


Fig. 1.7 *Example of overtraining: Letter data set.*

19,000 objects were used for testing. A quadratic discriminant classifier (QDC) from PRTOOLS is used.[3] We vary the regularization parameter r, $r \in [0, 1]$, which specifies to what extent we make use of the data. For $r = 0$ there is no regularization, we have more accuracy on the training data and less certainty that the classifier will perform well on unseen data. For $r = 1$, the classifier might be less accurate on the training data, but can be expected to perform at the same rate on unseen data. This dilemma can be transcribed into everyday language as "specific expertise" versus "common sense." If the classifier is trained to expertly recognize a certain data set, it might have this data-specific expertise and little common sense. This will show as high testing error. Conversely, if the classifier is trained to have good common sense, even if not overly successful on the training data, we might expect it to have common sense with any data set drawn from the same distribution.

In the experiment r was decreased for 20 steps, starting with $r_0 = 0.4$ and taking r_{k+1} to be $0.8 \times r_k$. Figure 1.7 shows the training and the testing errors for the 20 steps.

This example is intended to demonstrate the overtraining phenomenon in the process of varying a parameter, therefore we will look at the tendencies in the error curves. While the training error decreases steadily with r, the testing error decreases to a certain point, and then increases again. This increase indicates overtraining, that is, the classifier becomes too much of a data-specific expert and loses common sense. A common mistake in this case is to declare that the quadratic discriminant

[3] Discussed in Chapter 2.

TABLE 1.7 Comparison of Testing Errors of Two Classifiers.

	e_1	e_2	z	$\|z\| > 1.96$?	Outcome
Experiment 1	21.37	22.00	−2.11	Yes	Different ($e_1 < e_2$)
Experiment 2	23.38	22.00	4.54	Yes	Different ($e_1 > e_2$)

classifier has a testing error of 21.37 percent (the minimum in the bottom plot). The mistake is in that the *testing* set was used to find the best value of r.

Let us use the difference of proportions test for the errors of the classifiers. The testing error of our quadratic classifier (QDC) at the final 20th step (corresponding to the minimum training error) is 23.38 percent. Assume that the competing classifier has a testing error on this data set of 22.00 percent. Table 1.7 summarizes the results from two experiments. Experiment 1 compares the best testing error found for QDC, 21.37 percent, with the rival classifier's error of 22.00 percent. Experiment 2 compares the end error of 23.38 percent (corresponding to the minimum training error of QDC), with the 22.00 percent error. The testing data size in both experiments is $N_{ts} = 19{,}000$.

The results suggest that we would decide differently if we took the best testing error rather than the testing error corresponding to the best training error. Experiment 2 is the fair comparison in this case.

A point raised by Duin [16] is that the performance of a classifier depends upon the expertise and the *willingness* of the designer. There is not much to be done for classifiers with fixed structures and training procedures (called "automatic" classifiers in Ref. [16]). For classifiers with many training parameters, however, we can make them work or fail due to designer choices. Keeping in mind that there are no rules defining a fair comparison of classifiers, here are a few (non-Candide's) guidelines:

1. Pick the training procedures in advance and keep them fixed during training. When publishing, give enough detail so that the experiment is reproducible by other researchers.
2. Compare modified versions of classifiers with the *original* (nonmodified) classifier. For example, a distance-based modification of k-nearest neighbors (k-nn) should be compared with the standard k-nn first, and then with other classifier models, for example, neural networks. If a slight modification of a certain model is being compared with a totally different classifier, then it is not clear who deserves the credit, the modification or the original model itself.
3. Make sure that all the information about the data is utilized by all classifiers to the largest extent possible. For example, a clever initialization of a prototype-based classifier such as the learning vector quantization (LVQ) can make it favorite among a group of equivalent but randomly initialized prototype classifiers.

4. Make sure that the testing set has not been seen at any stage of the training.
5. If possible, give also the complexity of the classifier: training and running times, memory requirements, and so on.

1.5 BAYES DECISION THEORY

1.5.1 Probabilistic Framework

Although many types of uncertainty exist, the probabilistic model fits surprisingly well in most pattern recognition problems. We assume that the class label ω is a random variable taking values in the set of class labels $\Omega = \{\omega_1, \ldots, \omega_c\}$. The *prior probabilities*, $P(\omega_i)$, $i = 1, \ldots, c$, constitute the probability mass function of the variable ω,

$$0 \leq P(\omega_i) \leq 1$$

and

$$\sum_{i=1}^{c} P(\omega_i) = 1 \tag{1.28}$$

We can construct a classifier based on this information only. To make the smallest possible number of mislabelings, we should always label an object with the class of the highest prior probability.

However, by measuring the relevant characteristics of the objects, organized as the vector $\mathbf{x} \in \Re^n$, we should be able to make a more accurate decision about this particular object. Assume that the objects from class ω_i are distributed in $\Re^n$ according to the *class-conditional probability density function* (pdf) $p(\mathbf{x}|\omega_i)$, $p(\mathbf{x}|\omega_i) \geq 0$, $\forall \mathbf{x} \in \Re^n$, and

$$\int_{\Re^n} p(\mathbf{x}|\omega_i)\, d\mathbf{x} = 1, \qquad i = 1, \ldots, c \tag{1.29}$$

The likelihood of $\mathbf{x} \in \Re^n$ is given by the *unconditional* pdf

$$p(\mathbf{x}) = \sum_{i=1}^{c} P(\omega_i) p(\mathbf{x}|\omega_i) \tag{1.30}$$

Given the prior probabilities and the class-conditional pdfs we can calculate the *posterior probability* that the true class label of the measured $\mathbf{x}$ is ω_i using the Bayes formula

$$P(\omega_i|\mathbf{x}) = \frac{P(\omega_i) p(\mathbf{x}|\omega_i)}{p(\mathbf{x})} = \frac{P(\omega_i) p(\mathbf{x}|\omega_i)}{\sum_{j=1}^{c} P(\omega_j) p(\mathbf{x}|\omega_j)} \tag{1.31}$$

Equation (1.31) gives the probability mass function of the class label variable ω *for the observed* $\mathbf{x}$. The decision for that particular $\mathbf{x}$ should be made with respect to the posterior probability. Choosing the class with the highest posterior probability will lead to the smallest possible mistake when classifying $\mathbf{x}$.

The probability model described above is valid for the discrete case as well. Let $\mathbf{x}$ be a discrete variable with possible values in $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_s\}$. The only difference from the continuous-valued case is that instead of class-conditional pdf, we use *class-conditional probability mass functions* (pmf), $P(\mathbf{x}|\omega_i)$, giving the probability that a particular value from $\mathbf{V}$ occurs if we draw at random an object from class ω_i. For all pmfs,

$$0 \leq P(\mathbf{x}|\omega_i) \leq 1, \quad \forall \mathbf{x} \in \mathbf{V}, \qquad \text{and} \qquad \sum_{j=1}^{s} P(\mathbf{v}_j|\omega_i) = 1 \tag{1.32}$$

1.5.2 Normal Distribution

An important example of class-conditional pdf is the normal distribution denoted $p(\mathbf{x}|\omega_i) \sim N(\mu_i, \Sigma_i)$, where $\mu_i \in \Re^n$, and Σ_i are the parameters of the distribution. μ_i is the mean of class ω_i, and Σ_i is an $n \times n$ covariance matrix. The class-conditional pdf is calculated as

$$p(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^{n/2}\sqrt{|\Sigma_i|}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i)\right] \tag{1.33}$$

where $|\Sigma_i|$ is the determinant of Σ_i. For the one-dimensional case, $\mathbf{x}$ and μ_i are scalars, and Σ_i reduces to the variance of x for class ω_i, denoted σ_i^2. Equation (1.33) simplifies to

$$p(x|\omega_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{1}{2}\left(\frac{x - \mu_i}{\sigma_i}\right)^2\right] \tag{1.34}$$

The normal distribution (or also Gaussian distribution) is the most natural assumption reflecting the following situation: there is an "ideal prototype" of class ω_i (a point in $\Re^n$) and all class members are distorted versions of it. Small distortions are more likely to occur than large distortions, causing more objects to be located in the close vicinity of the ideal prototype than far away from it. The prototype is represented by the population mean μ_i and the scatter of the points around it is associated with the covariance matrix Σ_i.

Example: Data Cloud Shapes and the Corresponding Covariance Matrices. Figure 1.8 shows four two-dimensional data sets generated from normal distributions with different covariance matrices as displayed underneath the respective scatterplot.

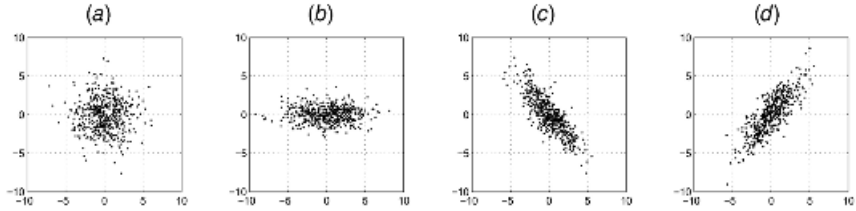


Fig. 1.8 *Normally distributed data sets with mean* $[0,0]^T$ *and different covariance matrices shown underneath.*

Plots (a) and (b) are generated with *independent* (noninteracting) features, that is, the data cloud is either spherical (subplot (a)), or stretched along the coordinate axes (subplot (b)). Notice that for these cases the off-diagonal entries of the covariance matrix are zeros. Subplots (c) and (d) represent cases where the features are *dependent*.

In the case of *independent features* we can decompose the n-dimensional pdf as a product of n one-dimensional pdfs. Let σ_{ik}^2 be the kth diagonal entry of the covariance matrix Σ_i and μ_{ik} be the kth component of μ_i. Then

$$p(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^{n/2}\sqrt{|\Sigma_i|}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i)\right]$$
$$= \prod_{k=1}^{n}\left\{\frac{1}{\sqrt{(2\pi)}\sigma_{ik}} \exp\left[-\frac{1}{2}\left(\frac{x_k-\mu_{ik}}{\sigma_{ik}}\right)^2\right]\right\} \tag{1.35}$$

The *cumulative distribution function* for a random variable $X \in \Re$ with a normal distribution, $\Phi(z) = P(X \leq z)$, is available in tabulated form from any statistical textbook.[4]

1.5.3 Generate Your Own Data

Trivial though it might be, sometimes you need a piece of code to generate your own data set with specified probabilistic characteristics.

[4] $\Phi(z)$ can be approximated with error at most 0.005 for $0 \leq z \leq 2.2$ as [25]

$$\Phi(z) = 0.5 + \frac{z(4.4-z)}{10}$$

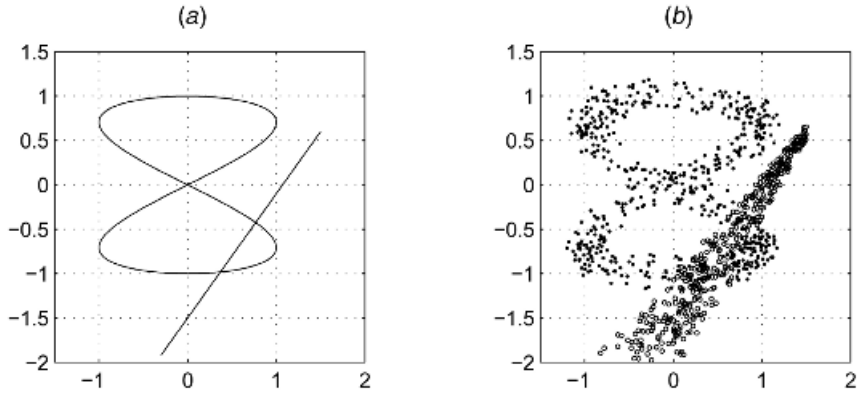


Fig. 1.9 *(a) The skeletons of the two classes to be generated. (b) The generated data set.*

1.5.3.1 Noisy Geometric Figures. The following example suggests one possible way for achieving this. (The Matlab code is shown in Appendix 1E.)

Suppose we want to generate two classes in $\Re^2$ with prior probabilities 0.6 and 0.4, respectively. Each class will be distributed along a piece of a parametric curve. Let class ω_1 have a skeleton of a *Lissajous figure* with a parameter t, such that

$$x = a \sin nt, \quad y = b \cos t, \qquad t \in [-\pi, \pi] \tag{1.36}$$

Pick $a = b = 1$ and $n = 2$. The Lissajous figure is shown in Figure 1.9*a*.

Let class ω_2 be shaped as a segment of a line with a parametric equation

$$x = t, \quad y = at + b, \qquad \text{for } t \in [-0.3, 1.5] \tag{1.37}$$

Let us pick $a = 1.4$ and $b = -1.5$. The segment is depicted in Figure 1.9*a*.

We shall draw random samples with uniform distributions along the skeletons with overlaid normal distributions of specified variances. For ω_1 we shall use $\sigma^2 = 0.005$ on both axes and a diagonal covariance matrix. For ω_2, we shall use $\sigma_1^2 = 0.01 \times (1.5 - x)^2$ and $\sigma_2^2 = 0.001$. We chose σ_1 to vary with x so that smaller x values exhibit larger variance. To design the data set, select the total number of data points T, and follow the list of steps below. The normal distributions for the example are generated within the code. Only the standard (uniform) random generator of Matlab will be used.

1. Generate a random number $r \in [0,1]$.
2. If $r < 0.6$, then proceed to generate a point from ω_1.
   (a) Generate randomly t in the interval $[-\pi, \pi]$.
   (b) Find the point (x, y) on the curve using Eq. (1.36).
   (c) To superimpose the noise generate a series of triples of random numbers u, v within $[-3\sigma, 3\sigma]$, and $w \in [0, 1]$, until the following condition,

coming from the multivariate normal distribution formula (1.33), is met

$$w < \frac{1}{2\pi\sigma^2}\exp\left[-\frac{1}{2}\left(\frac{u^2+v^2}{\sigma^2}\right)\right] \tag{1.38}$$

where $\sigma^2 = 0.005$.

(d) Add the new point $(x+u, y+v)$ and a label ω_1 for it to the data set.

3. Otherwise ($r \geq 0.6$) proceed to generate a point from ω_2.
   (a) Generate randomly t in the interval -0.3, 1.5.
   (b) Find the point (x, y) on the line using Eq. (1.37).
   (c) To superimpose the noise generate a series of triples of random numbers

$$\begin{aligned} u &\in [-3\sigma_1, 3\sigma_1] \\ v &\in [-3\sigma_2, 3\sigma_2] \\ w &\in [0, 1] \end{aligned}$$

until the following condition is met

$$w < \frac{1}{2\pi\sigma_1\sigma_2}\exp\left[-\frac{1}{2}\left(\frac{u^2}{\sigma_1^2}+\frac{v^2}{\sigma_2^2}\right)\right] \tag{1.39}$$

where $\sigma_1^2 = 0.01 \times (1.5 - x)^2$ and $\sigma_2^2 = 0.001$.

(d) Add the new point $(x+u, y+v)$ and a label ω_2 for it to the data set.

Any pdfs can be simulated in a similar way.

1.5.3.2 Rotated Check-Board Data. The Matlab code below generates a data set with complicated decision boundaries. The data is two-dimensional and spans the unit square $[0, 1] \times [0, 1]$. The classes are placed as the white and the black squares of a check-board and then the whole board is rotated at an angle α. A parameter "a" specifies the side of the individual squares. For example, if $a = 0.5$, then before rotation, there will be four squares in total. Figure 1.10 shows a data set of 100,000 points generated from the Matlab code for two sets of input parameters.

The properties that make this data set interesting for experimental purposes are:

- The two classes are perfectly separable, therefore zero error is the target for both training and testing.
- The classification regions for the same class are disjoint.
- The boundaries are not parallel to the coordinate axes.
- The classification performance will be highly dependent on the sample size.

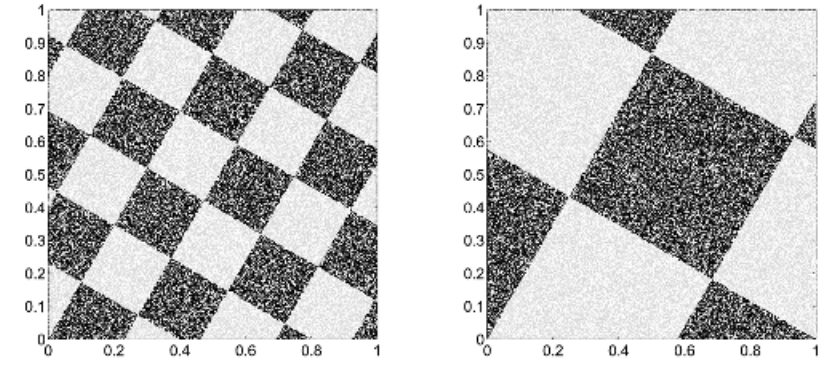


Fig. 1.10 *Rotated check-board data (100,000 points in each plot).*

The Matlab code for N data points is:

```
function [d,labd]=gendatcb(N,a,alpha);
d=rand(N,2);
d_transformed=[d(:,1)*cos(alpha)-d(:,2)*sin(alpha), ...
               d(:,1)*sin(alpha)+d(:,2)*cos(alpha)];
s=ceil(d_transformed(:,1)/a)+floor(d_transformed(:,2)/a);
labd=2-mod(s,2);
```

1.5.4 Discriminant Functions and Decision Boundaries

The class with the highest posterior probability is the most natural choice for a given $\mathbf{x}$. Therefore the posterior probabilities can be used directly as the discriminant functions, that is,

$$g_i(\mathbf{x}) = P(\omega_i|\mathbf{x}), \qquad i = 1, \ldots, c \tag{1.40}$$

Hence we rewrite the maximum membership rule (1.3) as

$$D(\mathbf{x}) = \omega_{i^*} \in \Omega \Leftrightarrow P(\omega_{i^*}|\mathbf{x}) = \max_{i=1,\ldots,c} \{P(\omega_i|\mathbf{x})\} \tag{1.41}$$

In fact, a set of discriminant functions leading to the same classification regions would be

$$g_i(\mathbf{x}) = P(\omega_i)p(\mathbf{x}|\omega_i), \qquad i = 1, \ldots, c \tag{1.42}$$

because the denominator of Eq. (1.31) is the same for all i, and so will not change the ranking order of g_i values. Another useful set of discriminant functions derived from the posterior probabilities is

$$g_i(\mathbf{x}) = \log[P(\omega_i)p(\mathbf{x}|\omega_i)], \qquad i = 1, \ldots, c \tag{1.43}$$

Example: Decision/Classification Boundaries. Let $x \in \Re$. Figure 1.11 shows two sets of discriminant functions for three normally distributed classes with

$$P(\omega_1) = 0.45, \qquad p(x|\omega_1) \sim N\big(4, (2.0)^2\big)$$
$$P(\omega_2) = 0.35, \qquad p(x|\omega_2) \sim N\big(5, (1.2)^2\big)$$
$$P(\omega_3) = 0.20, \qquad p(x|\omega_3) \sim N\big(7, (1.0)^2\big)$$

The first set (top plot) depicts a set of functions (1.42), $P(\omega_i)p(x|\omega_i)$, $i = 1, 2, 3$. The classification boundaries are marked with bullets on the x-axis. The posterior probabilities (1.40) are depicted in the bottom plot. The classification regions specified by the boundaries are displayed with different shades of gray in the bottom plot. Note that the same regions are found in both plots.

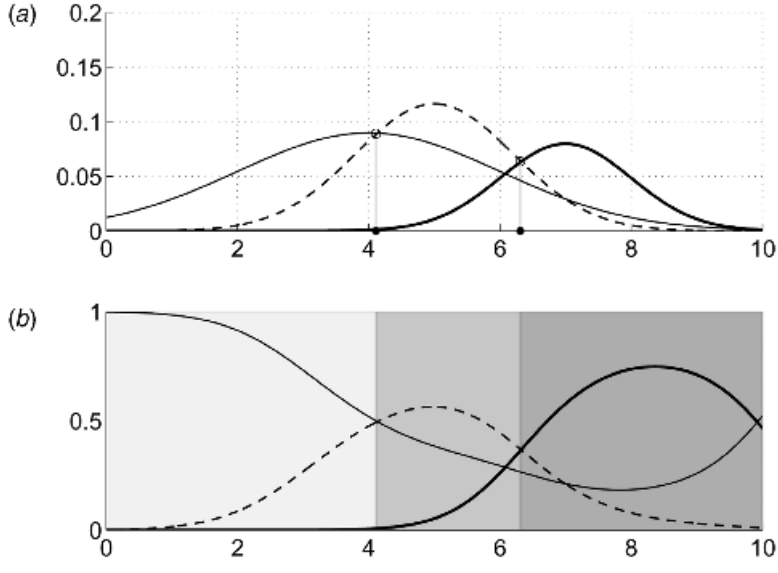


Fig. 1.11 *(a) Plot of two equivalent sets of discriminant functions: $P(\omega_1)p(x|\omega_1)$ (the thin line), $P(\omega_2)p(x|\omega_2)$ (the dashed line), and $P(\omega_3)p(x|\omega_3)$ (the thick line). (b) Plot of the three posterior probability functions $P(\omega_1|x)$ (the thin line), $P(\omega_2|x)$ (the dashed line), and $P(\omega_3|x)$ (the thick line). In both plots $x \in [0,\ 10]$.*

Sometimes more than two discriminant function might tie at the boundaries. Ties are resolved randomly.

1.5.5 Bayes Error

Let D^* be a classifier that always assigns the class label with the largest posterior probability. Since for every $\mathbf{x}$ we can only be correct with probability

$$P(\omega_{i^*}|\mathbf{x}) = \max_{i=1,\ldots,c} \{P(\omega_i|\mathbf{x})\} \tag{1.44}$$

there is some inevitable error. The overall probability of error of D^* is the sum of the errors of the individual $\mathbf{x}$s weighted by their likelihood values, $p(\mathbf{x})$; that is,

$$P_e(D^*) = \int_{\Re^n} [1 - P(\omega_{i^*}|\mathbf{x})]\, p(\mathbf{x})\, d\mathbf{x} \tag{1.45}$$

It is convenient to split the integral into c integrals, one on each classification region. For this case class ω_{i^*} will be specified by the region's label. Then

$$P_e(D^*) = \sum_{i=1}^{c} \int_{\mathcal{R}_i^*} [1 - P(\omega_i|\mathbf{x})]\, p(\mathbf{x})\, d\mathbf{x} \tag{1.46}$$

where $\mathcal{R}_i^*$ is the classification region for class ω_i, $\mathcal{R}_i^* \cap \mathcal{R}_j^* = \emptyset$ for any $j \neq i$ and $\bigcup_{i=1}^{c} \mathcal{R}_i^* = \Re^n$. Substituting Eq. (1.31) into Eq. (1.46) and taking into account that $\sum_{i=1}^{c} \int_{\mathcal{R}_i^*} \cdot = \int_{\Re^n} \cdot$,

$$P_e(D^*) = \sum_{i=1}^{c} \int_{\mathcal{R}_i^*} \left[1 - \frac{P(\omega_i)p(\mathbf{x}|\omega_i)}{p(\mathbf{x})}\right] p(\mathbf{x})\, d\mathbf{x} \tag{1.47}$$

$$= \int_{\Re^n} p(\mathbf{x})\, d\mathbf{x} - \sum_{i=1}^{c} \int_{\mathcal{R}_i^*} P(\omega_i)p(\mathbf{x}|\omega_i)\, d\mathbf{x} \tag{1.48}$$

$$= 1 - \sum_{i=1}^{c} \int_{\mathcal{R}_i^*} P(\omega_i)p(\mathbf{x}|\omega_i)\, d\mathbf{x} \tag{1.49}$$

Note that $P_e(D^*) = 1 - P_c(D^*)$, where $P_c(D^*)$ is the overall probability of correct classification of D^*.

Consider a different classifier, D, which produces classification regions $\mathcal{R}_1, \ldots, \mathcal{R}_c$, $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ for any $j \neq i$ and $\bigcup_{i=1}^{c} \mathcal{R}_i = \Re^n$. Regardless of the way the regions are formed, the error of D is

$$P_e(D) = \sum_{i=1}^{c} \int_{\mathcal{R}_i} [1 - P(\omega_i|\mathbf{x})]\, p(\mathbf{x})\, d\mathbf{x} \tag{1.50}$$

The error of D^* is the smallest possible error, called the *Bayes error*. The example below illustrates this concept.

Example: Bayes Error. Consider the simple case of $x \in \Re$ and $\Omega = \{\omega_1, \omega_2\}$. Figure 1.12 displays the discriminant functions in the form $g_i(\mathbf{x}) = P(\omega_i)p(\mathbf{x}|\omega_i)$, $i = 1, 2$, $x \in [0, 10]$.

For two classes,

$$P(\omega_1|x) = 1 - P(\omega_2|x) \tag{1.51}$$

and $P_e(D^*)$ in Eq. (1.46) becomes

$$P_e(D^*) = \int_{\mathcal{R}_1^*} [1 - P(\omega_1|\mathbf{x})]\, p(\mathbf{x})\, d\mathbf{x} + \int_{\mathcal{R}_2^*} [1 - P(\omega_2|\mathbf{x})]\, p(\mathbf{x})\, d\mathbf{x} \tag{1.52}$$

$$= \int_{\mathcal{R}_1^*} P(\omega_2|\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} + \int_{\mathcal{R}_2^*} P(\omega_1|\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{1.53}$$

$$= \int_{\mathcal{R}_1^*} P(\omega_2) p(\mathbf{x}|\omega_2)\, d\mathbf{x} + \int_{\mathcal{R}_2^*} P(\omega_1) p(\mathbf{x}|\omega_1)\, d\mathbf{x} \tag{1.54}$$

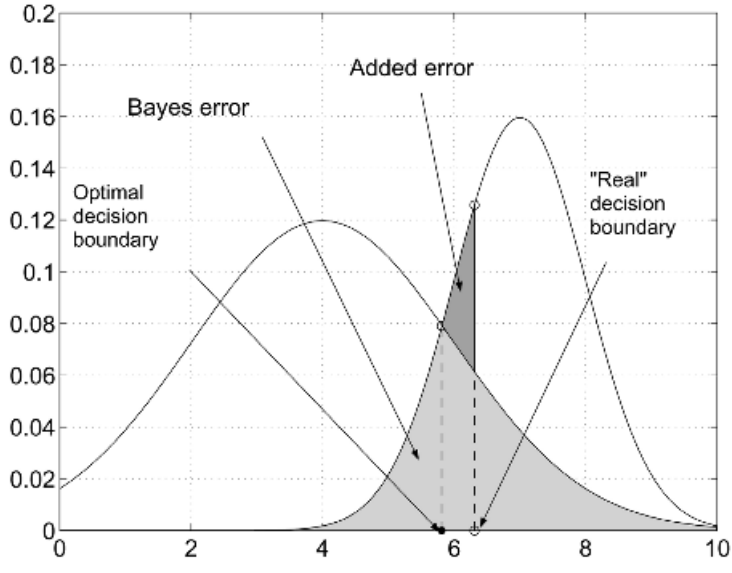


Fig. 1.12 *Plot of two discriminant functions $P(\omega_1)p(x|\omega_1)$ (left curve) and $P(\omega_2)p(x|\omega_2)$ (right curve) for $x \in [0, 10]$. The light gray area corresponds to the Bayes error, incurred if the optimal decision boundary (denoted by •) is used. The dark gray area corresponds to the additional error when another boundary (denoted by ○) is used.*

By design, the classification regions of D^* correspond to the true highest posterior probabilities. The bullet on the x-axis in Figure 1.12 splits $\Re$ into $\mathcal{R}_1^*$ (to the left) and $\mathcal{R}_2^*$ (to the right). According to Eq. (1.54), the Bayes error will be the area under $P(\omega_2)p(\mathbf{x}|\omega_2)$ in $\mathcal{R}_1^*$ plus the area under $P(\omega_1)p(\mathbf{x}|\omega_1)$ in $\mathcal{R}_2^*$. The total area corresponding to the Bayes error is marked in light gray. If the boundary is shifted to the left or right, additional error will be incurred. We can think of this boundary as the result from classifier D, which is an imperfect approximation of D^*. The shifted boundary, depicted by an open circle, is called in this example the "real" boundary. Region $\mathcal{R}_1$ is therefore $\mathcal{R}_1^*$ extended to the right. The error calculated through Eq. (1.54) is the area under $P(\omega_2)p(\mathbf{x}|\omega_2)$ in the whole of $\mathcal{R}_1$, and extra error will be incurred, measured by the area shaded in dark gray. Therefore, using the *true* posterior probabilities or an equivalent set of discriminant functions guarantees the smallest possible error rate, called the *Bayes error*.

Since the true probabilities are never available in practice, it is impossible to calculate the exact Bayes error or design the perfect Bayes classifier. Even if the probabilities were given, it will be difficult to find the classification regions in $\Re^n$ and calculate the integrals. Therefore, we rely on estimates of the error as discussed in Section 1.3.

1.5.6 Multinomial Selection Procedure for Comparing Classifiers

Alsing et al. [26] propose a different view of classification performance. The classifiers are compared on a labeled data set, relative to each other in order to identify which classifier has most often been closest to the true class label. We assume that each classifier gives at its output a set of c posterior probabilities, one for each class, guessing the chance of that class being the true label for the input vector $\mathbf{x}$. Since we use labeled data, the posterior probabilities for the *correct* label of $\mathbf{x}$ are sorted and the classifier with the largest probability is nominated as the winner for this $\mathbf{x}$.

Suppose we have classifiers $D_1, \ldots, D_L$ to be compared on a data set $\mathbf{Z}$ of size N. The *multinomial selection procedure* consists of the following steps.

1. For $i = 1, \ldots, c$,
   (a) Use only the N_i data points whose true label is ω_i. Initialize an $N_i \times L$ performance array T.
   (b) For every point $\mathbf{z}_j$, such that $l(\mathbf{z}_j) = \omega_i$, find the estimates of the posterior probability $P(\omega_i|\mathbf{z}_j)$ guessed by each classifier. Identify the largest posterior probability, store a value of 1 for the winning classifier D_q by setting $T(j, q) = 1$ and values 0 for the remaining $L - 1$ classifiers, $T(j, k) = 0$, $k = 1, \ldots, L$, $k \neq q$.
   (c) Calculate an estimate of each classifier being the winner for class ω_i assuming that the number of winnings follows a binomial distribution. The estimate of this probability will be the total number of 1s stored

for this classifier divided by the number of tests.

$$\hat{P}(D_k \text{ wins } |\omega_i) = \frac{1}{N_i} \sum_{j=1}^{N_i} T(j, k)$$

2. End i.
3. To find an overall measure of the performance of classifier D_k, use the sum of its performance values for the classes weighted by the respective prior probabilities or their estimates; that is,

$$\hat{P}(D_k \text{ wins}) = \sum_{i=1}^{c} \hat{P}(D_k \text{ wins}|\omega_i)\hat{P}(\omega_i) \tag{1.55}$$

If we estimate the prior probabilities as the proportions on $\mathbf{Z}$, then the estimate of the probability $P(D_k \text{ wins})$ becomes

$$\hat{P}(D_k \text{ wins}) = \frac{1}{N} \sum_{i=1}^{c} N_i \hat{P}(D_k \text{ wins } |\omega_i) \tag{1.56}$$

Multinomial selection procedure has been demonstrated in Ref. [26] to be very sensitive in picking out the winner, unlike the traditional error-based comparisons.

1.6 TAXONOMY OF CLASSIFIER DESIGN METHODS

Since in real-life problems we usually do not know the true prior probabilities nor the class-conditional pdfs, we can only design flawed versions of the Bayes classifier. Statistical pattern recognition provides a variety of classifier models [1,2,4,7,10,27–29]. Figure 1.13 shows one possible taxonomy of methods for classifier design. The boxes contain representative classifier models from the respective categories. Some of the classifier models are detailed in the next chapter.

One solution is to try to estimate $P(\omega_i)$ and $p(\mathbf{x}|\omega_i)$, $i = 1, \ldots, c$, from $\mathbf{Z}$ and substitute the estimates $\hat{P}(\omega_i)$ and $\hat{p}(\mathbf{x}|\omega_i)$ in the discriminant functions $g_i(\mathbf{x}) = P(\omega_i)p(\mathbf{x}|\omega_i)$, $i = 1, \ldots, c$. This is called the *plug-in* approach to classifier design. Approximating $p(\mathbf{x}|\omega_i)$ as a function of $\mathbf{x}$ divides classifier design methods into two big groups: *parametric* and *non parametric*. On the other side of the diagram are classifier design methods that are not derived by approximating the pdfs but rather by devising decision boundaries or discriminant functions empirically.

The distinction between the groups is not clear-cut. For example, *radial basis function* (RBF) networks from the group of structural approximation of the discrimi-

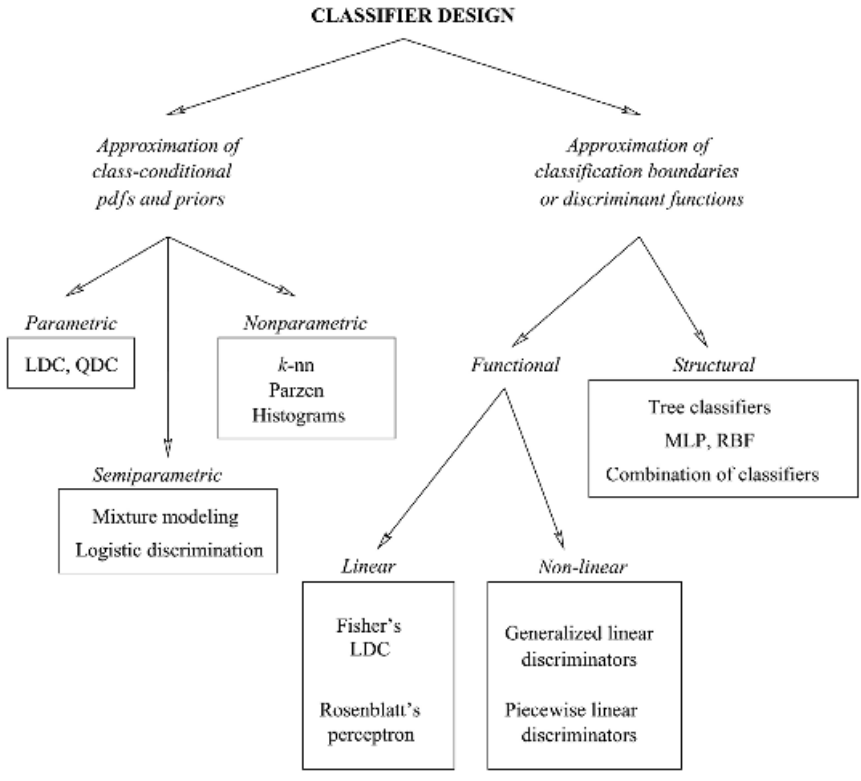


Fig. 1.13 *A taxonomy of methods for classifier design.*

nant functions can be moved to the group of functional approximation, or even to the group of semiparametric pdf modeling [30]. Similarly, the k-nearest neighbor (k-nn) method, although theoretically linked with the nonparametric pdf estimation, produces a direct estimate of the discriminant functions and can be put under the heading of structural designs for approximating the discriminant functions.

There is no consensus on a single taxonomy, or even about the definition of parametric and nonparametric classifiers. Lippmann [31] lists five types of classifiers:

- probabilistic (LDC, QDC, Parzen);
- global (multilayer perceptron (MLP));
- local (radial basis function neural networks (RBF));
- nearest-neighbor type (k-nn, learning vector quantization neural networks (LVQ));
- rule-forming (binary decision trees, rule-based systems).

Holmström et al. [32] consider another grouping:

- classifiers based on density estimation:
  - parametric (LDC, QDC);
  - nonparametric (k-nn, kernel methods, finite mixtures, RBF).
- classifiers based on regression:
  - parametric (linear regression, logistic regression, MLP);
  - nonparametric (projection pursuit, additive models).
- other classifiers (e.g., prototype-based: LVQ, k-nn for small k)

Some authors distinguish between neural and nonneural classifiers, local and global classifiers, and so on.

1.7 CLUSTERING

Clustering aims at finding groups in data. "Cluster" is an intuitive concept and does not have a mathematically rigorous definition. The members of one cluster should be similar to one another and dissimilar to the members of other clusters. A clustering algorithm operates on an unlabeled data set $\mathbf{Z}$ and produces a *partition* on it, denoted $P = (\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(c)})$, where $\mathbf{Z}^{(i)} \subseteq \mathbf{Z}$ and

$$\mathbf{Z}^{(i)} \cap \mathbf{Z}^{(j)} = \emptyset, \qquad i, j = 1, \ldots, c, \; i \neq j \tag{1.57}$$

$$\bigcup_{i=1}^{c} \mathbf{Z}^{(i)} = \mathbf{Z} \tag{1.58}$$

There is a vast amount of literature on clustering [33–37] looking for answers to the main questions, among which are:

- Is there really a structure in the data or are we imposing one by our clustering algorithms?
- How many clusters should we be looking for?
- How do we define similarity between objects in the feature space?
- How do we know whether our clustering results are good?

Two main groups of clustering algorithms are *hierarchical clustering* (agglomerative and divisive) and *nonhierarchical clustering*. The nearest neighbor (single linkage) clustering algorithm shown in Figure 1.14 is an example of the hierarchical group whereas the c-means clustering algorithm (called also "k-means") shown in Figure 1.15 is an example of the nonhierarchical group.

Single linkage clustering

1. Pick the number of clusters c and a similarity measure $\mathcal{S}(a, b)$ between two objects a and b. Initialize the procedure by defining an individual cluster for each point in $\mathbf{Z}$.
2. Identify the two most similar clusters and join them as a new cluster, dismissing the initial two clusters. The similarity between clusters A and B is measured as
$$\min_{a \in A, b \in B} \mathcal{S}(a, b).$$
3. Repeat step 2 until c clusters are found.

Fig. 1.14 *The single linkage clustering algorithm.*

***c*-Means clustering**

1. Pick the number of clusters c and a similarity measure $\mathcal{S}(a, b)$ between two objects a and b. Initialize the c cluster centers (e.g., by randomly selecting c points from $\mathbf{Z}$ to be the centers).
2. Label all points in $\mathbf{Z}$ with respect to their similarity to the cluster centers: each point is assigned to the cluster with the most similar center.
3. Calculate the new centers as the mean of the points from $\mathbf{Z}$ assigned to the respective cluster.
4. Repeat steps 2 and 3 until no change in the centers occurs.

Fig. 1.15 *The c-means clustering algorithm.*

Example: Single Linkage and *c*-Means Clustering for the Banana Data. Consider a banana data set generated with $\sigma = 0.7$, with 50 points on each banana shape as shown in Figure 1.16.

Figure 1.17*a* shows the results from running the single linkage clustering for the banana data and Figure 1.17*b* shows the results from running c-means clustering on the same data set. The obtained clusters are denoted by dots and $\times$'s. Ideally, the two clusters should match the original labels. The points that are mislabeled (i.e., generated by banana "A" and classed as banana "B") are encircled in both figures. Neither of the two algorithms was able to identify the two bananas. Finding touching clusters appears to be difficult for most clustering algorithms.

Fig. 1.16 *Banana data for the clustering example.*

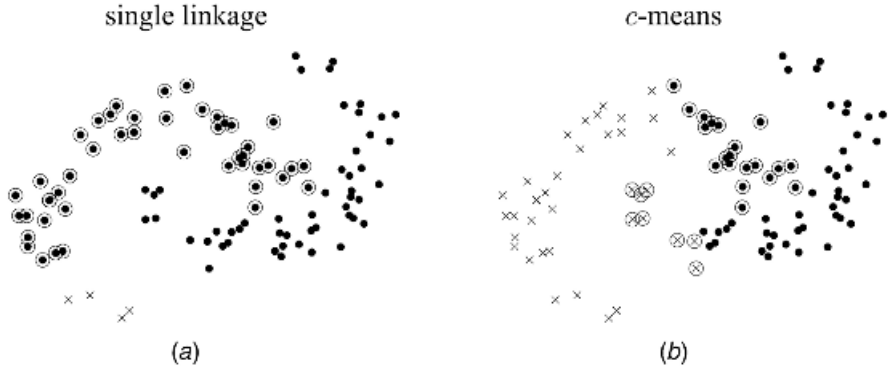


Fig. 1.17 *Results from the single linkage* (*a*) *and c-means* (*b*) *clustering on a banana data set with 50 points on each banana shape. The "mislabeled" points are encircled.*

APPENDIX 1A *K*-HOLD-OUT PAIRED *t*-TEST

The Matlab code for the calculation of the t-statistic for the banana data and two classifiers (9-nn and Parzen) is given below. We assume that the data set $\mathbf{Z}$ has already been loaded into the memory, organized in an $N \times n$ matrix named "Z," where N is the number of objects and n is the number of features. Another matrix named "labZ" of size $N \times 1$ must contain the class labels of $\mathbf{Z}$ encoded as the consecutive integers $1, 2, \ldots, c$. For simplicity we assumed that N is even. The functions from the toolbox PRTOOLS are underlined

```
% Input Z, labZ, number of points N
K=30; % Number of runs
for i=1:K
    iii=randperm(N); % random permutation of 1...N
    ZZ=Z(iii,:);     % arrange the data
    labZZ=labZ(iii); % shuffle the labels accordingly
    t1=ZZ(1:(N/2),:);t1l=labZZ(1:(N/2));
                     % take the first half for training
    t2=ZZ((N/2)+1:N,:);t2l=labZZ((N/2)+1:N);
                     % take the second half for testing
    e1=testk(t1,t1l,9,t2,t2l);
                     % test the 9-nn classifier
                     % (no training is necessary here)
    P1(i)=1-e1; % store the testing accuracy
    W=parzenc(t1,t1l);  % train the Parzen classifier
    e2=testd(W,t2,t2l); % test the Parzen classifier
    P2(i)=1-e2;   % store the testing accuracy
end

P=P1-P2; % calculate the K differences
barP=mean(P) % calculate and print the mean difference
sumD=sum((P-barP).^2);
t=barP*sqrt(K)/sqrt(sumD/(K-1))
                     % calculate and print the t-statistic
```

APPENDIX 1B *K*-FOLD CROSS-VALIDATION PAIRED *t*-TEST

All the input variables are as in Appendix 1A. Here it is recommended that N is a multiple of K so that exactly the same size training and testing data is used in each of the K splits.

```
% Input Z, labZ, number of points N
K=10; % Number of partitions
n1=ceil(N/K);   % find the size of the testing data sets
last=N-(K-1)*n1;% find the size of the last set (if any)
if last==0,
   last=n1;     % K divides N, all pieces are the same
end
if last<n1/2,   % if the last piece is smaller than
                % half of the size of the others,
                % then issue a warning
    fprintf('%s\n','Warning: imbalanced testing sets')
end
v=[]; % construct indicator-labels for the K subsets
for i=1:K-1;
   v=[v;ones(n1,1)*i];
end
v=[v;ones(last,1)*K];
```

```
for i=1:K
    l=v==i;
    t1=Z(~l,:);    % training data
    t1l=labZ(~l);  % training labels
    t2=Z(l,:);     % testing data
    t2l=labZ(l);   % testing labels
    e1=testk(t1,t1l,9,t2,t2l); % 9-nn
    P1(i)=1-e1;
    W=parzenc(t1,t1l); % Parzen classifier
    e2=testd(W,t2,t2l);
    P2(i)=1-e2;
end
P=P1-P2; % calculate the K differences
barP=mean(P) % calculate and print the mean difference
sumD=sum((P-barP).^2);
t=barP*sqrt(K)/sqrt(sumD/(K-1))
                    % calculate and print the t-statistic
```

APPENDIX 1C 5 × 2cv PAIRED *t*-TEST

```
% Input Z, labZ, number of points N
K=5;
for i=1:K,
    iii=randperm(N);
    ZZ=Z(iii,:); % shuffle the data
    labZZ=labZ(iii);
    % Split into halves
    t1=ZZ(1:(N/2),:);t1l=labZZ(1:(N/2));
    t2=ZZ((N/2)+1:N,:);t2l=labZZ((N/2)+1:N);
    e1=testk(t1,t1l,9,t2,t2l); % 9-nn
    P1(i)=1-e1;
    W=parzenc(t1,t1l); % Parzen classifier
    e2=testd(W,t2,t2l);
    P2(i)=1-e2;
    % Swap training and testing
    e3=testk(t2,t2l,9,t1,t1l); % 9-nn
    P3(i)=1-e3;
    W=parzenc(t2,t2l); % Parzen classifier
    e4=testd(W,t1,t1l);
    P4(i)=1-e4;
end;

D1=P1-P2;D2=P3-P4; % the difference arrays
D=(D1+D2)/2 % the five averaged differences
s2=(D1-D).^2+(D2-D).^2 % the 5 variances
tildeT=D1(1)/sqrt(mean(s2))
           % calculate and plot the t-statistic
```

APPENDIX 1D 500 GENERATIONS OF TRAINING/TESTING DATA AND CALCULATION OF THE PAIRED *t*-TEST STATISTIC

```
K=500;
for i=1:K,
    [t1,dummy]=gendatb(50,50,1.5); % generate training data
    t1l=[zeros(50,1);ones(50,1)];
    [t2,dummy]=gendatb(50,50,1.5); % generate testing data
    t2l=[zeros(50,1);ones(50,1)];
    e1=testk(t1,t1l,9,t2,t2l); % 9-nn
    P1indep(i)=1-e1;
    W=parzenc(t1,t1l); % parzen classifier
    e2=testd(W,t2,t2l);
    P2indep(i)=1-e2;
end;

Pindep=P1indep'-P2indep';% calculate the K differences
barPindep=mean(Pindep) % the mean difference
sumDindep=sum((Pindep-barPindep).^2) % $s^2$
t=barPindep*sqrt(K)/sqrt(sumDindep/(K-1))
                      % calculate and print the t-statistic
```

APPENDIX 1E DATA GENERATION: LISSAJOUS FIGURE DATA

The code below generates the data shown in Figure 1.9. The steps are explained in the text. We note that the code has not been designed to optimize the execution time. We kept the assignments and calculations of some constants within the loops for clarity and readability.

```
T=1000;
s=0.005; % sigma^2 for class omega_1
DataSet=[];

for i=1:T
    if rand<0.6
        % Generate a point from class omega_1
        % First, generate a point on the curve
        t=(rand-0.5)*2*pi; % pick t in [-pi,pi]
        x=sin(2*t);y=cos(t);
        % Superimpose a normal distribution
        flag=0; % we don't have the noise coordinates yet
        while flag == 0,
            u=(rand-0.5)*6*sqrt(s);
            v=(rand-0.5)*6*sqrt(s);
            w=rand;
            if w<((1/(2*pi*s))*exp(-(u^2+v^2)/(2*s))),
                flag=1;
                % u and v are suitable noise coordinates
            end % if w
```

```
        end % while
        DataSet=[DataSet;x+u,y+v]; % Add to the data set
        Labels(i)=1; % Store the label
    else
        % Generate a point from class omega_2
        % First, generate a point on the curve
        t=-0.3+rand*1.8; % pick t in [-0.3, 1.5]
        x=t;y=1.4*t-1.5;
        s1=0.01*(1.5-x)^2;s2=0.001; % variances
        % Superimpose a normal distribution
        flag=0; % we don't have the noise coordinates yet
        while flag == 0,
            u=(rand-0.5)*6*sqrt(s2);
            v=(rand-0.5)*6*sqrt(s1);
            w=rand;
            if w<((1/(2*pi*sqrt(s1*s2)))*exp(-(u^2/s2+v^2/s1)/2)),
                flag=1;
                % u and v are suitable noise coordinates
            end % if w
        end % while
        DataSet=[DataSet;x+u,y+v]; % Add to the data set
        Labels(i)=2; % Store the label
    end % if rand
end % for i
```

2

Base Classifiers

2.1 LINEAR AND QUADRATIC CLASSIFIERS

Linear and quadratic classifiers are named after the type of discriminant functions they use. Thus any set of linear functions $g_i : \Re^n \to \Re$, $i = 1, \ldots, c$,

$$g_i(\mathbf{x}) = w_{i0} + \mathbf{w}_i^T \mathbf{x}, \qquad \mathbf{x}, \mathbf{w}_i \in \Re^n, \quad w_{i0} \in \Re \tag{2.1}$$

can be thought of as a linear classifier.

2.1.1 Linear Discriminant Classifier

Training of linear classifiers has been rigorously studied in the early pattern recognition literature [2], dating back to the Fisher's linear discriminant, 1936 [312]. Below we derive the linear classifier as the minimum-error (Bayes) classifier for normally distributed classes with equal covariance matrices. We shall call this model the *linear discriminant classifier* (LDC). LDC is simple to calculate from data and is reasonably robust, i.e., the results might be surprisingly good even when the classes do not have normal distributions.

As discussed in the previous chapter, any set of discriminant functions obtained by a monotonic transformation from the posterior probabilities $P(\omega_i|\mathbf{x})$ constitutes an optimal set in terms of minimum error. We form such a set by taking

$$g_i(\mathbf{x}) = \log[P(\omega_i)p(\mathbf{x}|\omega_i)], \qquad i = 1, \ldots, c \tag{2.2}$$

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

where $P(\omega_i)$ is the prior probability for class ω_i and $p(\mathbf{x}|\omega_i)$ is the class-conditional probability density function (pdf). Suppose that all classes are normally distributed with means μ_i and covariance matrices Σ_i, that is, $p(\mathbf{x}|\omega_i) \sim N(\mu_i, \Sigma_i)$, $i = 1, \ldots, c$. Then Eq. (2.2) takes the form

$$\begin{aligned} g_i(\mathbf{x}) &= \log[P(\omega_i)] + \log\left\{\frac{1}{(2\pi)^{n/2}\sqrt{|\Sigma_i|}}\exp\left[-\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i)\right]\right\} \\ &= \log[P(\omega_i)] - \frac{n}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma_i|) \\ &\quad - \frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i) \end{aligned} \tag{2.3}$$

Assume that all class-covariance matrices are the same, that is, $\Sigma_i = \Sigma$, and $p(\mathbf{x}|\omega_i) \sim N(\mu_i, \Sigma)$. Opening the parentheses in the last term of Eq. (2.3) and discarding all terms that do not depend on ω_i, we obtain a new set of discriminant functions

$$g_i(\mathbf{x}) = \log[P(\omega_i)] - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + \mu_i^T\Sigma^{-1}\mathbf{x} = w_{i0} + \mathbf{w}_i^T\mathbf{x} \tag{2.4}$$

where $w_{i0} \in \Re$ and $\mathbf{w}_i \in \Re^n$ are the coefficients of the linear discriminant function g_i.

In reality, the classes are neither normally distributed nor are the true values of μ_i and Σ_i known. We can still calculate the coefficients w_{i0} and $\mathbf{w}_i$ from data using estimates of the means and the covariance matrices but the obtained classifier will not be equivalent to the minimum-error (Bayes) classifier.

2.1.2 Quadratic Discriminant Classifier

Assume that the classes are normally distributed, but now with class-specific covariance matrices, that is, $p(\mathbf{x}|\omega_i) \sim N(\mu_i, \Sigma_i)$. The set of optimal discriminant functions is obtained from Eq. (2.3) by discarding all terms that do not depend on the class label ω_i,

$$g_i(\mathbf{x}) = w_{i0} + \mathbf{w}_i^T\mathbf{x} + \mathbf{x}^T W_i\mathbf{x} \tag{2.5}$$

where

$$w_{i0} = \log[P(\omega_i)] - \frac{1}{2}\mu_i^T\Sigma_i^{-1}\mu_i - \frac{1}{2}\log(|\Sigma_i|) \tag{2.6}$$

$$\mathbf{w}_i = \Sigma_i^{-1}\mu_i \tag{2.7}$$

and

$$W_i = -\frac{1}{2}\Sigma_i^{-1} \tag{2.8}$$

The estimates of the parameters for LDC and the quadratic discriminant classifier (QDC) are calculated from data. Let N_i be the number of objects in our data set $\mathbf{Z}$ from class ω_i, $i = 1, \ldots, c$, and $l(\mathbf{z}_j) \in \Omega$ be the class label of $\mathbf{z}_j \in \mathbf{Z}$. The means are obtained by

$$\hat{\boldsymbol{\mu}}_i = \frac{1}{N_i} \sum_{l(\mathbf{z}_j)=\omega_i} \mathbf{z}_j \tag{2.9}$$

and the covariance matrices, by[5]

$$\hat{\boldsymbol{\Sigma}}_i = \frac{1}{N_i} \sum_{l(\mathbf{z}_j)=\omega_i} (\mathbf{z}_j - \hat{\boldsymbol{\mu}}_i)(\mathbf{z}_j - \hat{\boldsymbol{\mu}}_i)^T \tag{2.10}$$

The common covariance matrix for LDC is obtained as the weighted average of the separately estimated class-conditional covariance matrices.

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{i=1}^{c} N_i \Sigma_i \tag{2.11}$$

2.1.3 Using Data Weights with a Linear Discriminant Classifier and Quadratic Discriminant Classifier

For the purposes of designing ensembles of classifiers it is important to have a mechanism to incorporate data weights into LDC and QDC. The most natural way for that is to calculate the weighted $\hat{\boldsymbol{\mu}}_i$ and $\hat{\boldsymbol{\Sigma}}_i$. Let $W(j)$ be the weight of object $\mathbf{z}_j \in \mathbf{Z}$, W^i be the sum of the weights of all objects in $\mathbf{Z}$ from ω_i, and $W = \sum_{i=1}^{c} W^i$ be the total sum of weights for $\mathbf{Z}$. Then

$$\hat{\boldsymbol{\mu}}_i^{(w)} = \frac{1}{W^i} \sum_{l(\mathbf{z}_j)=\omega_i} W(j)\mathbf{z}_j \tag{2.12}$$

and

$$\hat{\boldsymbol{\Sigma}}_i^{(w)} = \frac{1}{W^i} \sum_{l(\mathbf{z}_j)=\omega_i} W(j)(\mathbf{z}_j - \hat{\boldsymbol{\mu}}_i)(\mathbf{z}_j - \hat{\boldsymbol{\mu}}_i)^T \tag{2.13}$$

[5] We use the maximum likelihood estimate of the covariance matrices and note that this estimate is biased. For an unbiased estimate take $\hat{\boldsymbol{\Sigma}}_i = 1/(N_i - 1) \sum_{l(\mathbf{z}_j)=\omega_i} (\mathbf{z}_j - \hat{\boldsymbol{\mu}}_i)(\mathbf{z}_j - \hat{\boldsymbol{\mu}}_i)^T$.

For the common covariance matrix for LDC,

$$\hat{\Sigma}^{(w)} = \frac{\sum_{i=1}^{c} W^i \Sigma_i}{W} \tag{2.14}$$

2.1.4 Regularized Discriminant Analysis

Since $\hat{\Sigma}$ (for LDC) or $\hat{\Sigma}_i$ (for QDC) have to be inverted, it is important that these matrices are not singular or close to singular. This often poses a problem, especially for small data sizes (small N) and high dimensionality (large n). When N is smaller than the number of parameters that need to be estimated, some of the parameters are not identifiable from the data and the problem is said to be *ill-posed*. When N only marginally exceeds the number of parameters to be estimated, the problem is called *poorly posed*. To overcome this, we can use *regularization*.

Regularized discriminant analysis (RDA) is proposed by Friedman [38] to cope with ill- and poorly posed problems. We shall take Friedman's definition of regularization as an "attempt to improve the estimates by biasing them away from their sample-based values toward values that are deemed to be more 'physically plausible' ".

Two ways of stabilizing the estimates $\hat{\Sigma}_i$ are considered. First, averaging of $\hat{\Sigma}_i$ weighted by the number of observations giving Σ, Eq. (2.11), is already a good regularization move. However, by using one Σ for all classes we reduce QDC to LDC. We can quantify the amount of this reduction by introducing a parameter, $\lambda \in [0, 1]$, and use

$$\hat{\Sigma}_i(\lambda) = (1-\lambda)\hat{\Sigma}_i + \lambda\hat{\Sigma} \tag{2.15}$$

Friedman [38] uses the weighted estimates

$$\hat{\Sigma}_i^{(w)}(\lambda) = \frac{(1-\lambda)W^i\hat{\Sigma}_i^{(w)} + \lambda W\hat{\Sigma}^{(w)}}{(1-\lambda)W^i + \lambda W} \tag{2.16}$$

In both formulas, $\lambda = 1$ means that QDC becomes LDC because all the classes share the same covariance matrix $\hat{\Sigma}$, and $\lambda = 0$ means that no regularization is implemented. Thus, λ spanning the interval between 0 and 1 produces a family of classifiers between LDC and QDC.

The problem might be ill-posed or poorly posed even for the LDC. The second way of regularizing the estimates is to add a term to $\hat{\Sigma}_i$ that will shrink it toward a multiple of the identity matrix

$$\hat{\Sigma}_i(r) = (1-r)\hat{\Sigma}_i + \frac{r}{n}\mathrm{tr}(\hat{\Sigma}_i)I \tag{2.17}$$

where tr denotes the trace of the matrix and I denotes the identity matrix of size $n \times n$. This estimate has the effect of "equalizing" the eigenvalues of $\hat{\Sigma}_i$ which

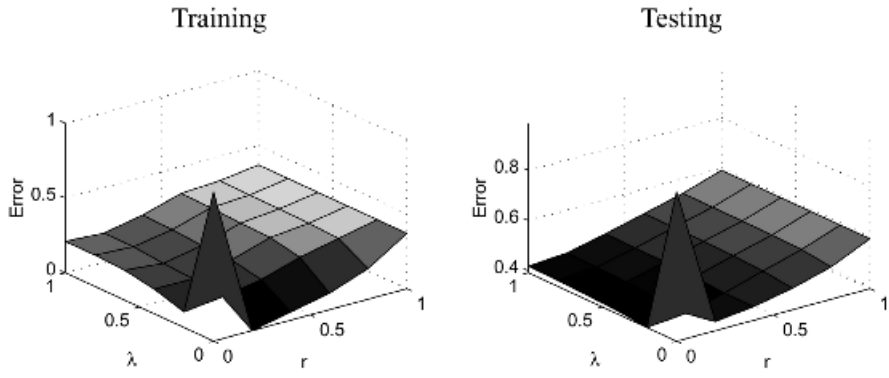


Fig. 2.1 *Training and testing error rates for pairs of values* (λ, r) *for RDA with the Letters data set* (*see Figure 1.7*).

counters the bias inherent in sample-based estimates of the eigenvalues [38]. The parameter $r \in [0, 1]$ determines to what extent we want to equalize the eigenvalues. For $r = 0$, there is no regularization and for $r = 1$, $\hat{\Sigma}_i$ is a diagonal matrix with eigenvalues equal to the averaged eigenvalues of the sample-based estimate of the covariance matrix.

Recall the example from Figure 1.7, where a regularization parameter was varied. In this example we regularized the covariance matrices using Eq. (2.17) for 20 different values of r.[6]

Friedman defines RDA to be a combination of the two regularization ways so that

$$\hat{\Sigma}_i(\lambda, r) = (1 - r)\hat{\Sigma}_i(\lambda) + \frac{r}{n}\text{tr}[\hat{\Sigma}_i(\lambda)]I \tag{2.18}$$

The values of the two parameters, λ and r, have to be determined from the data. Depending on the problem, a different pair of values might be preferred. A cross-validation procedure is suggested in Ref. [38] for selecting from a grid of pairs (λ, r).

Example: Choosing Parameter Values for Regularized Discriminant Analysis. To simulate a small data set, we used the first 200 objects from the Letter data for training and the remaining 19,800 objects for testing. Since there are 26 classes and 16 features, it is likely that the sample estimates of the covariance matrices will benefit from regularizing. Figure 2.1 shows the training and testing surfaces over the unit square $[0, 1]^2$ spanned by (λ, r).

There were singular covariance matrices for $\lambda = 0$ and $r = 0$, which is the cause of the high peak of the error around the origin. A very small regularization appears to be sufficient to make the most of the linear–quadratic compromise model. The training error for $\lambda = 0$, $r = 0.2$ appeared to be 0, and the corresponding testing

[6] The QDC implementation in the toolbox PRTOOLS allows for r to be specified as an external parameter.

error was 42.8 percent. The best testing error, 39.2 percent was found for $\lambda = 0.2$, $r = 0$, which suggests a model not very far from the QDC. As the figure shows, further regularization would increase the error, leading eventually to the 37.0 percent training error and 59.1 percent testing error of LDA trained on this data set.

2.2 NONPARAMETRIC CLASSIFIERS

The three classifier models in the previous section assumed that the classes are normally distributed. If this is true, then theoretically either LDC or QDC is the optimal classifier for the problem. Most often we do not have sufficient knowledge of the underlying distributions. We can still use LDC or QDC (or a model in-between provided by RDA) regardless of the suspected nonoptimality. Practice shows that LDC and QDC often yield reasonable accuracy.

In nonparametric designs the probability density function $p(\mathbf{x}|\omega_i)$ is estimated in the vicinity of $\mathbf{x}$ in $\Re^n$. The probability p^* that $\mathbf{x}$ is in a certain region $R \subset \Re^n$ is

$$p^* = P(\mathbf{x} \in R) = \int_R p(\mathbf{u})\, d\mathbf{u} \tag{2.19}$$

Assume that N samples are drawn from the distribution in question. Using the binomial distribution with parameters (p^*, N), the probability that exactly k samples fall in R is given by

$$p_k = \binom{N}{k} (p^*)^k (1 - p^*)^{N-k} \tag{2.20}$$

and the value p^* can be estimated as the proportion of the points in R with respect to the total number of samples; that is,

$$p^* \approx \frac{k}{N} \tag{2.21}$$

Let $\mathbf{x}$ be a point inside region R. For a small R, assuming that $p(\mathbf{u})$ is approximately constant in R, p^* can also be approximated as

$$p^* \approx p(\mathbf{x}) \int_R d\mathbf{u} = p(\mathbf{x}) V_R \tag{2.22}$$

where V_R is the volume of R in $\Re^n$. Joining Eqs. (2.21) and (2.22), and solving for $p(\mathbf{x})$ we obtain

$$p(\mathbf{x}) \approx \frac{k}{N V_R} \tag{2.23}$$

When N tends to infinity and the region R shrinks to a point ($V_R \to 0$), Eq. (2.23) produces the exact $p(\mathbf{x})$. The above result is an important point of departure of various nonparametric classification methods.

2.2.1 Multinomial Classifiers

To derive the multinomial classifier (we will call it also "the method of histograms") from Eq. (2.23), we fix N and V, and calculate k from $\mathbf{Z}$. The feature space is divided into bins (cells). Consider a bin B containing m points from the data set $\mathbf{Z}$. Let m_i be the number of points in B from class ω_i, $i = 1, \ldots, c$, $m = m_1 + \cdots + m_c$. According to Eq. (2.23)

$$p(\mathbf{x}) \approx \frac{m}{NV_B} \tag{2.24}$$

where V_B is the volume of the bin, and

$$p(\mathbf{x}|\omega_i) \approx \frac{m_i}{N_i V_B} \tag{2.25}$$

Here N_i is the number of elements from class ω_i in $\mathbf{Z}$. The prior probabilities are usually estimated by

$$\hat{P}(\omega_i) = \frac{N_i}{N} \tag{2.26}$$

Then the posterior probabilities are obtained as

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} \approx \frac{\dfrac{m_i}{N_i V_B}\dfrac{N_i}{N}}{\dfrac{m}{NV_B}} \tag{2.27}$$

hence

$$P(\omega_i|\mathbf{x}) \approx \frac{m_i}{m} \tag{2.28}$$

The approximation of the Bayes classifier is obtained by using the approximated values of the posterior probabilities for the classes as the discriminant functions.[7] Thus, the bin B is labeled according to the largest m_i in it, and all points in B get the same class label. To classify an input $\mathbf{x}$ (not in $\mathbf{Z}$), the histogram classifier finds the bin where $\mathbf{x}$ belongs and retrieves the bin's class label. Practically, the histogram classifier is a look-up table (usually a big one) whose accuracy depends on the number of bins and the sample size N.

A drawback of the histogram classifier is that the total number of bins grows exponentially with the dimensionality of the problem, n. If we use M bins per axis, the total number of bins will be M^n. This requires correspondingly large

[7] In Chapter 5 we discuss a smoothed estimate of this probability by the Laplace estimator.

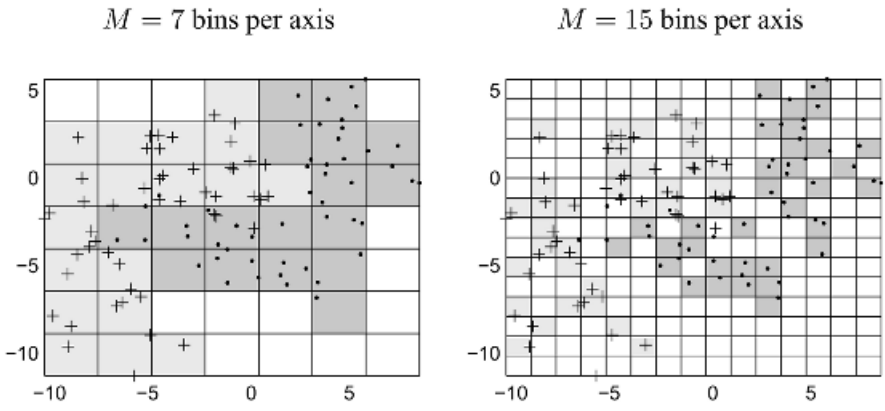


Fig. 2.2 *Classification regions found by a histogram classifier on the banana data. Plotted also is the training set.*

data sets because otherwise many bins will be empty or may contain too few elements to provide reliable estimates of the posterior probabilities. This phenomenon is known as the *curse of dimensionality*. The effect of the small number of data points can be alleviated using the Laplace correction as explained in Chapter 5.

Example: Multinomial Classifier for the Banana Data. We applied the multinomial classifier on the training set of the banana data ($N = 200$). The results for $M = 7$ and $M = 15$ bins per axis are shown in Figure 2.2.

The bins where the number of elements for both classes tie, or that are empty, are left white while the others are shaded. The gray level indicates the class labels. We calculated the classification error of this design using the leave-one-out method. For each $\mathbf{z}_j \in \mathbf{Z}$ we found its bin, and then recalculated the class label of the bin by updating m_1 or m_2. For example, assume that bin B originally contained $m_1 = 4$ points from ω_1 and $m_2 = 3$ points from ω_2 (so, B is labeled in ω_1). Let $\mathbf{z}_B$ be a point from $\mathbf{Z}$ such that it is located in B, and its class label is $l(\mathbf{z}_B) = \omega_1$. The updated values for the leave-one-out (in this case leave $\mathbf{z}_B$ out) are $m_1 \leftarrow m_1 - 1 = 3$ and $m_2 \leftarrow m_2 = 3$. Now the class label of B is obtained by breaking the tie randomly. If $\mathbf{z}_B$ was from class ω_2, the label assigned to it would be ω_1 because the updated values in that case would be $m_1 \leftarrow m_1 = 4$ and $m_2 \leftarrow m_2 - 1 = 2$. We also calculated the resubstitution error and the testing error on the independent test set. Table 2.1 shows these errors.

TABLE 2.1 Error Rates (in %) for Two Multinomial Classifiers for the Banana Data.

M	Resubstitution	Leave-one-out	Testing
7	7	20	10
15	5	35	26

With 15 bins the classification boundary is more precise but a larger part of the feature space remains uncovered by the classifier (the white regions). This means that more observations will have to be classified randomly and indeed the error on the independent testing set for $M = 15$ is bigger than the error for $M = 7$. The leave-one-out estimate seems overpessimistic in both examples. To examine this further, we ran the histogram classifier for $M = 2, 3, \ldots, 30$ and calculated the three errors as before. Figure 2.3 depicts the error rates versus M.

The curse of dimensionality leads to quick overtraining as the opposite trends of the training and the testing errors indicate. The plot also demonstrates that the leave-one-out estimate of the testing accuracy tends to be overpessimistic.

The multinomial classifier is both time and memory consuming, and, besides, critically depends on how large the data set is. The number of bins acts as a smoothing factor: the more bins we have, the noisier the approximation of the discriminant function. With large M, it is more likely that large regions of the feature space will not be covered. The method of histograms is hardly ever used in the form described here. However, is it an intuitive nonparametric model related to rule-based classifiers and especially fuzzy classifiers (cf. [39]).

Fig. 2.3 *Resubstitution, leave-one-out and testing error rates of the histogram classifier versus the number of bins per axis.*

2.2.2 Parzen Classifier

This model is based again on Eq. (2.23) where N and V_R are fixed, and k is found from the data. Let $K(\mathbf{t})$, $\mathbf{t} = [t_1, \ldots, t_n]^T \in \Re^n$, be a *kernel function* (or *Parzen window*), which peaks at the origin, is nonnegative, and has integral one over $\Re^n$. The simplest model of a Parzen window is the following function

$$K(\mathbf{t}) = \begin{cases} 1, & \text{if } |t_i| \leq \dfrac{1}{2}, \forall i = 1, \ldots, n \\ 0, & \text{otherwise} \end{cases} \tag{2.29}$$

This function defines a hyperbox in $\Re^n$ with side 1, centered at the origin. For all points $\mathbf{t}$ within the hypercube, $K(\mathbf{t}) = 1$ and for points in $\Re^n$ outside the hypercube, $K(\mathbf{t}) = 0$. Let the Parzen window be centered at some $\mathbf{x}$. Using the set $\mathbf{Z}$ for calculating Eq. (2.23), and taking into account that $V_R = 1$, the pdf at $\mathbf{x}$ can be approximated as

$$p(\mathbf{x}) \approx \frac{k}{N} = \frac{1}{N} \sum_{j=1}^{N} K(\mathbf{Z}_j - \mathbf{x}) \tag{2.30}$$

This formula can be interpreted in a different way if we assume that there are N hyperboxes, each centered on one point from $\mathbf{Z}$. The approximation is then calculated as the proportion of such hyperboxes that contain $\mathbf{x}$. The multidimensional kernel function centered on $\mathbf{z}_j \in \Re^n$ is usually expressed in the form $(1/h^n)K(\mathbf{x} - \mathbf{z}_j/h)$, where h is a smoothing parameter and

$$\int_{\Re^n} \frac{1}{h^n} K\Big(\frac{\mathbf{x} - \mathbf{z}_j}{h}\Big)\, d\mathbf{x} = 1 \tag{2.31}$$

A common choice of the kernel function is the multidimensional Gaussian kernel [4]:

$$\frac{1}{h^n}\, K_G\Big(\frac{\mathbf{x} - \mathbf{z}_k}{h}\Big) = \frac{1}{h^n\, (2\pi)^{n/2} \sqrt{|S|}} \exp\left[-\frac{1}{2h^2} (\mathbf{x} - \mathbf{z}_k)^T S^{-1} (\mathbf{x} - \mathbf{z}_k) \right] \tag{2.32}$$

Here S is a specified covariance matrix determining the shape of the kernel. The class-conditional pdfs are estimated using the sample set $\mathbf{Z}$ [4,24] by

$$\hat{p}(\mathbf{x}|\omega_i) = \frac{1}{N_i} \sum_{l(\mathbf{z}_j) = \omega_i} \frac{1}{h^n} K\Big(\frac{\mathbf{x} - \mathbf{z}_j}{h}\Big) \tag{2.33}$$

where N_i is the number of elements of $\mathbf{Z}$ from class ω_i. The estimate is asymptotically unbiased if the smoothing parameter h is a function of the number of samples N_i, such

that [4]

$$\lim_{N_i \to \infty} h(N_i) = 0 \tag{2.34}$$

Taking Eq. (2.26) as the estimates of the prior probabilities we obtain

$$\hat{P}(\omega_i|\mathbf{x}) = \frac{1}{Np(\mathbf{x})} \sum_{l(\mathbf{z}_j)=\omega_i} \frac{1}{h^n} K\left(\frac{\mathbf{x}-\mathbf{z}_j}{h}\right) \tag{2.35}$$

$$= C(\mathbf{x}, h, N) \sum_{l(\mathbf{z}_j)=\omega_i} K\left(\frac{\mathbf{x}-\mathbf{z}_j}{h}\right) \tag{2.36}$$

where the term $C(\mathbf{x}, h, N)$ depends on $\mathbf{x}$, h, and N, but not on the class label. The approximation of the conditional pdfs under the above conditions is asymptotically unbiased. Therefore, using the class-conditional pdfs estimated by Eq. (2.36) we obtain asymptotically the Bayes classifier. A set of optimal discriminant functions can be obtained from Eq. (2.36) by ignoring $C(\mathbf{x}, h, N)$; that is,

$$g_i(\mathbf{x}) = \sum_{l(\mathbf{z}_j)=\omega_i} K\left(\frac{\mathbf{x}-\mathbf{z}_j}{h}\right) \tag{2.37}$$

For the Gaussian kernel (2.32)

$$g_i(\mathbf{x}) = \sum_{l(\mathbf{z}_j)=\omega_i} \exp\left\{-\frac{1}{2h^2}[d_M(\mathbf{x}, \mathbf{z}_j)]^2\right\} \tag{2.38}$$

where

$$[d_M(\mathbf{x}, \mathbf{z}_j)]^2 = (\mathbf{x}-\mathbf{z}_j)^T S^{-1} (\mathbf{x}-\mathbf{z}_j) \tag{2.39}$$

is the squared *Mahalanobis distance* between $\mathbf{x}$ and $\mathbf{z}_j$ in $\Re^n$.

Parzen classifier is a beautiful theoretical model whose main disadvantages are:

- Parzen classifier needs all of $\mathbf{Z}$ as the prototype set, which can be too time-consuming for large N.
- The choice of h is difficult: small h leads to spiky approximations of the pdfs; big h oversmooths the pdfs.

A practical recommendation offered by some authors is to try several values of h and select the one producing the smallest error rate. *Reduced* and *weighted* Parzen models are proposed in the literature [32,40]. The real value of Parzen classifier lies in the fact that it is the statistical counterpart of several important classification methods such as radial basis function networks [41,42], the probabilistic neural network (PNN) [43], and a number of fuzzy classifiers [44–46].

2.3 THE *k*-NEAREST NEIGHBOR RULE

Nonparametric classification is often associated with the notion of *prototype*. We can think of a prototype as a representative element from a class. The class label assigned to an example is based on the similarity of this example to one or more prototypes. Typically, similarity is defined in a geometrical sense, that is, based on a certain distance. The smaller the distance, the higher the similarity between $\mathbf{x}$ and the prototype.

The histogram classifier can be viewed as a prototype classifier. We can place a prototype at the geometrical center of each bin. The input $\mathbf{x}$ is considered similar to the prototype in whose bin it is located. The class label of that bin becomes the class label of $\mathbf{x}$. Parzen classifier can be regarded as a prototype classifier as well. Here the prototypes are all the elements of $\mathbf{Z}$. The votes for each class are weighed according to the distance between $\mathbf{x}$ and the prototype, and added up. The scores for the classes are then compared to find the class with the maximal support (largest discriminant function). The classical example of a prototype classifier, however, is the *k-nearest neighbor classifier* (k-nn).

2.3.1 Theoretical Background

k-nn is one of the most theoretically elegant and simple classification techniques [2,4,47]. Let $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_v\}$ be a labeled *reference set* containing v points in $\Re^n$, referred to as *prototypes*. The prototypes are labeled in the c classes; that is, for any $\mathbf{v}_i \in \mathbf{V}$, we know its class label $l(\mathbf{v}_i) \in \Omega$. In the classic nearest neighbor design, $\mathbf{V}$ is the whole of $\mathbf{Z}$. To classify an input $\mathbf{x}$, the k nearest prototypes are retrieved from $\mathbf{V}$ together with their class labels. The input $\mathbf{x}$ is labeled to the most represented class label amongst the k nearest neighbors.

To arrive at this classification method, we fix k and N in Eq. (2.23) and allow for a variable V_R. Assuming Euclidean distance, let R be the hypersphere containing exactly k of the elements of the reference set $\mathbf{V}$. The unconditional pdf is approximated as

$$p(\mathbf{x}) \approx \frac{k}{NV_R} \tag{2.40}$$

Denoting by k_i the number of elements in R from class ω_i, the class-conditional pdf for ω_i, $i = 1, \ldots, c$, approximated in R, is

$$p(\mathbf{x}|\omega_i) \approx \frac{k_i}{N_i V_R} \tag{2.41}$$

where N_i is the number of elements from class ω_i in $\mathbf{Z}$. Using Eq. (2.26) for estimating $P(\omega_i)$, the posterior probabilities are obtained as

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} \approx \frac{\frac{k_i}{N_i V_R}\frac{N_i}{N}}{\frac{k}{NV_R}} \tag{2.42}$$

hence

$$P(\omega_i|\mathbf{x}) \approx \frac{k_i}{k} \tag{2.43}$$

The minimum error (Bayes) classifier using the approximations above will assign $\mathbf{x}$ to the class with the highest posterior probability, that is, the class most represented amongst the k nearest neighbors of $\mathbf{x}$. The region R and the volume V_R, respectively, are specific for each $\mathbf{x} \in \Re^n$ and a data set $\mathbf{Z}$. The k-nn classification rule, however, assigns the class label using only the numbers k_i, $i = 1, \ldots, c$, so the winning label does not depend on V_R.

k-nn is Bayes-optimal when $N \to \infty$ and $V_R \to 0$. The expression $V_R \to 0$ is equivalent to $k \to \infty$ and $k/N \to 0$. That is, the error rate of the k-nn classifier, $P_{k\text{-}nn}$ satisfies

$$\lim_{\substack{N\to\infty \\ k\to\infty \\ k/N\to 0}} P_{k\text{-}nn} = P_B \tag{2.44}$$

where P_B is the Bayes error. When k is 1 (the *nearest neighbor rule*, denoted 1-nn), and $N \to \infty$, the error rate $P_{1\text{-}nn}$ is bounded from above by twice the Bayes error rate [2]; that is,

$$P_{1\text{-}nn} \leq 2P_B \tag{2.45}$$

Notice that different metrics will define different regions R. Regardless of the metric used, the k-nn rule is applied in the same way. The type of the metric does not change the asymptotic properties of k-nn because the shape of R is not fixed in Eq. (2.23).

Example: 1-nn, 3-nn and Voronoi Diagrams. Figure 2.4*a* displays the 1-nn and 3-nn rule on a randomly generated data set with two classes. The diamond shows the vector to be classified and the arrows join this vector to its three nearest neighbors. The new object is assigned to the class of "dots" by the 1-nn rule because such is the label of the closest neighbor. The 3-nn rule labels the object as a "cross" because such are the labels of the second and the third neighbors (majority: 2 out of 3 votes). The classification regions obtained by the 1-nn rule can be depicted

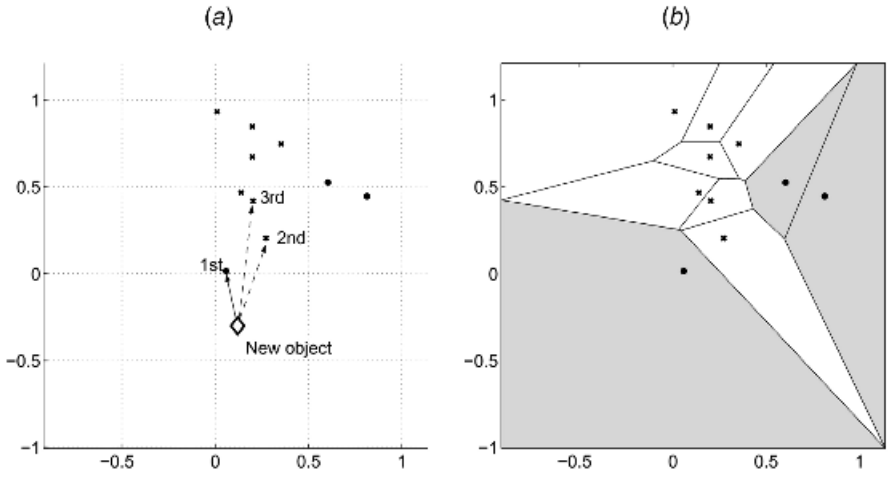


Fig. 2.4 *Illustration of the 1-nn and 3-nn classification rules and Voronoi diagrams.*

using *Voronoi diagrams* as shown in Figure 2.4*b*. The Voronoi bin V for $\mathbf{z}_j \in \mathbf{Z}$ is defined as the set of points in $\Re^n$ whose nearest neighbor from $\mathbf{Z}$ is $\mathbf{z}_j$; that is,

$$V(\mathbf{z}_j) = \left\{ \mathbf{x} \middle| \mathbf{x} \in \Re^n, d(\mathbf{z}_j, \mathbf{x}) = \min_{\mathbf{z}_k \in \mathbf{Z}} d(\mathbf{z}_k, \mathbf{x}) \right\} \tag{2.46}$$

where $d(\cdot, \cdot)$ is a distance in $\Re^n$. In this example we used the Euclidean distance.

There are three basic ways to vary the k-nn classifier:

- different distance metric in $\Re^n$;
- different value of k;
- edited versions of $\mathbf{Z}$ as prototype sets $\mathbf{V}$.

While there is a neat theory for the case of continuous-valued features, the problems with discrete and qualitative features start as early as defining similarity. Aha et al. [48] use the following similarity function between two objects represented by n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ with possibly mixed-type features.

$$\text{Similarity } (\mathbf{x}, \mathbf{y}) = -\sqrt{\sum_{i=1}^{n} f(x_i, y_i)} \tag{2.47}$$

where

$$f(x_i, y_i) = \begin{cases} (x_i - y_i)^2, & \text{if the } i\text{th feature is numeric,} \\ 1 - \mathcal{I}(x_i, y_i), & \text{if the } i\text{th feature is binary or symbolic.} \end{cases} \tag{2.48}$$

Recall that $\mathcal{I}(a, b)$ is an indicator function such that $\mathcal{I}(a, b) = 1$ if $a = b$, and 0 if $a \neq b$. Missing values are assumed to be maximally different from the present values. If both x_i and y_i are missing, then $f(x_i, y_i) = 1$.

Like the histogram method, k-nn is slow and memory-consuming. Unlike the histogram method, however, k-nn covers the whole feature space (e.g., see the Voronoi diagrams). The main drawbacks of k-nn methods are summarized in Ref. [48]:

- they are computationally expensive;
- they are intolerant to noise in the features;
- they are intolerant to redundant features;
- they are sensitive to the choice of the similarity function;
- there is no natural similarity measure for nominal-valued features;
- they do not have a mechanism for handling missing values;
- they provide little information regarding the structure of the data.

2.3.2 Finding *k*-nn Prototypes

Here we give methods to construct the k-nn reference set $\mathbf{V}$.

Let us assume that in the data set $\mathbf{Z}$, there are no identical objects (coinciding points in $\Re^n$) with different class labels. Then the 1-nn classifier with $\mathbf{Z}$ as the reference set is an ideal classifier producing no resubstitution errors. Each object will find itself as the nearest neighbor within $\mathbf{Z}$, thereby obtaining its own (correct) class label. Finding a smaller set of prototypes is essential for two reasons. The first one is computational: with a smaller number of prototypes, less time and memory resources will be required for the operation of the classifier. Reducing the computational demand is important for handling the massive data sets that are currently available in many application domains: industry, finance, medicine, biology, and so on. Secondly, in the process of finding $\mathbf{V}$, some noisy objects from $\mathbf{Z}$ could be filtered out [49] and so the classification accuracy can be improved.

To reduce the number of prototypes, either a subset of $\mathbf{Z}$ is selected (*prototype selection*) or a set of new prototypes is created (*prototype extraction*). Prototype selection is mostly related to the 1-nn classifier while prototype extraction is related to a class of neural networks, viz. the vector quantization models, LVQ. In both cases the idea is the same: *Use as few as possible prototypes to achieve as high as possible 1-nn (k-nn) accuracy.*

2.3.2.1 Edited k-nn (Prototype Selection). The simplest way to find prototypes is to select them from $\mathbf{Z}$. This is called *editing* of the reference set. There are two basic strategies [47]:

- Find a subset $\mathbf{V} \subseteq \mathbf{Z}$ that produces zero *resubstitution* errors on $\mathbf{Z}$ when used as a reference set with the nearest neighbor (1-nn) classifier. Such subsets are called *consistent*, and the techniques searching for the smallest consistent sub-

set are called *condensing* techniques. Typically, condensing techniques retain points from $\mathbf{Z}$ that are most likely to be misclassified (e.g., points that are usually close to the classification boundaries).

- Find a subset $\mathbf{V} \in \mathbf{Z}$ that has low (not necessarily zero) resubstitution error rate and *generalizes* well. This will be called *error-based editing* (in Ref. [1] it is called just "editing"). Error-based editing methods tend to eliminate the points close to the boundaries and retain those that belong "most certainly" in their own Bayes classification regions.

Being diametrically opposite, the two strategies typically lead to very different reference sets $\mathbf{V}$. Owing to their design, condensing methods cannot trade accuracy for reduction in the number of prototypes. Such trade-off is very desirable because at the expense of one or two misclassifications we might be able to halve the size of $\mathbf{V}$. Standard error-based editing methods do not have an explicit mechanism to limit the number of prototypes or penalize reference sets of high cardinality. They often produce more prototypes than are needed. Therefore it is reasonable to apply first an error-based method and then a condensing method on the resultant set of prototypes.

An alternative approach is to specify explicitly the number of prototypes and the amount of accuracy to trade off for reducing the number of prototypes. For example, we can use a criterion of the following type:

$$\max_{S \subseteq \mathbf{Z}} J(S) = \max_{S \subseteq \mathbf{Z}} \left\{ \text{Accuracy}(S) - \alpha \frac{|S|}{|\mathbf{Z}|} \right\} \tag{2.49}$$

where α is a constant weighting the importance of the cardinality reduction. The accuracy of the candidate subset S is measured using 1-nn with S as the reference set. We can use random search or guided random search through the subsets of $\mathbf{Z}$ to optimize $J(S)$. Possible optimization techniques are genetic algorithms [50–52] or tabu search [53].

Three editing algorithms are explained below. These were chosen because they are possibly the simplest, have different flavors, and have been the points of departure for many variations.

Hart's Method. This method belongs to the condensing group [54]. It gradually builds the subset $\mathbf{V}$ starting with $\mathbf{z}_1$ and moving into $\mathbf{V}$ every element of $\mathbf{Z}$ that is misclassified by 1-nn when using the current $\mathbf{V}$ as the reference set. The procedure loops through $\mathbf{Z}$ until all of $\mathbf{Z}$ is classified correctly. This technique tends to retain the points close to the classification boundaries and discard the inside points.

Wilson's Method. This method [55] is from the error-editing group. Wilson proposes to run k-nn (recommended value $k = 3$) on $\mathbf{Z}$ and mark for deletion all elements that are misclassified. By deleting the marked elements from $\mathbf{Z}$, we obtain the reference set $\mathbf{V}$ to be used with the 1-nn classifier. Wilson's method is the basis of the asymptotically optimal editing method called MULTIEDIT [1]. MULTIEDIT

works well on large data sets with cloud-shaped classes. If the overlap of the classes is high, MULTIEDIT could eventually rule out all elements of some of the classes [29,50].

Random Editing. This is simple and sometimes surprisingly successful [52,56]. We specify the desired number of prototypes $v = |\mathbf{V}|$, $c \leq v < N$ and the maximum number of trials T, generate T random subset-candidates $\mathbf{V} \subseteq \mathbf{Z}$, and return the set with the smallest resubstitution error.

Example: Prototype Selection for the Banana Data. The three basic editing methods were applied to the training part of the banana data. We also applied Wilson followed by Hart (denoted Wilson + Hart). The results are displayed in Figure 2.5.

The classification region of the "left" banana, obtained by the 1-nn classifier *using the edited reference set* is shaded in gray. The original training data is also plotted and the prototypes that the respective method has retained are encircled.

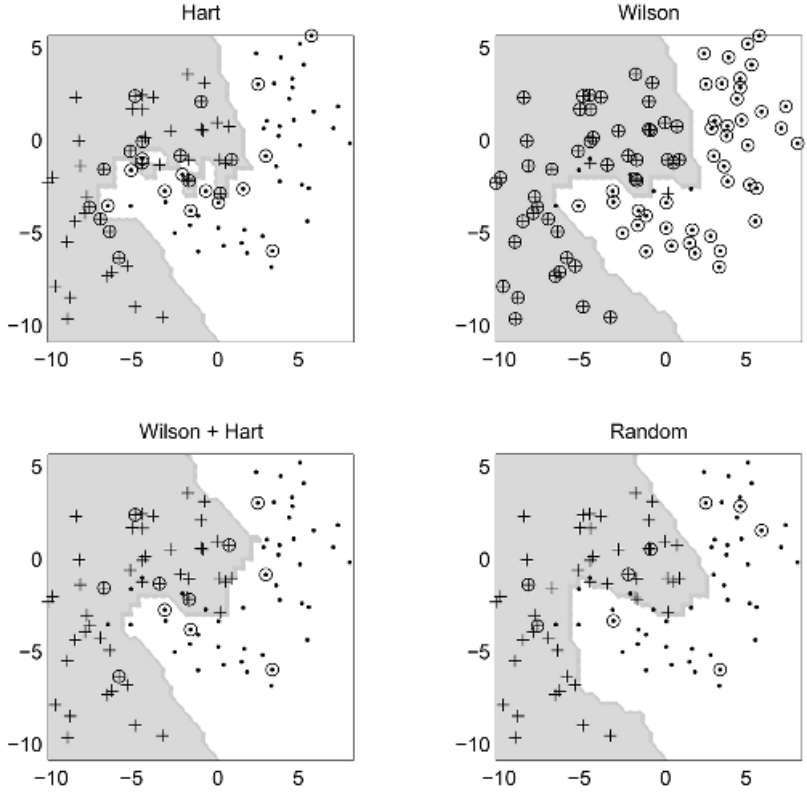


Fig. 2.5 *Training data and classification regions for the banana data set obtained by the 1-nn classifier and four prototype selection methods. The prototypes retained by the respective method are encircled.*

TABLE 2.2 Error Rates (in %) for 1-nn and Four Editing Methods for the Banana Data.

	Whole set	Hart	Wilson	Wilson + Hart	Random
Training error (%)	0	0	7	5	4
Testing error (%)	10	10	8	8	6
No. of prototypes	100	28	92	13	10

As expected, since the classes are not heavily overlapped, Wilson's method kept unnecessarily almost the whole data set. Hart's method retained mostly borderline points to preserve the boundary. Visually, the combination of the two (Wilson + Hart) improves on both. The random editing was run for $T = 1000$ and desired cardinality of the reference set $v = 10$. Table 2.2 shows the training and testing errors of the four methods.

We might have been lucky with the random editing for this example as it happened to be the best (let alone the fastest) among the four methods. It gave the lowest testing error with the smallest number of prototypes retained. This method is interesting in the context of multiple classifier systems because of its potential to discover disparate subsets of good reference quality as a basis for a diverse ensemble. Diversity, as we shall discuss later, is among the most important factors for the success of a classifier ensemble.

2.3.2.2 Calculating Prototypes from Data (Prototype Extraction). By *selecting* prototypes from $\mathbf{Z}$ we have a limited choice of points in $\Re^n$. Better results might be achieved by constructing $\mathbf{V} \subset \Re^n$ by choosing from the whole of $\Re^n$. There are numerous strategies and techniques for extracting prototypes including

Competitive Learning. Examples of this group are the neural network models called vector quantization (VQ) and learning vector quantization (LVQ) [57,58], and various modifications [59–63]. There are also competitive clustering algorithms such as the dog–rabbit algorithm, and Chang's method. A modification of the original Chang's method is proposed by Bezdek et al. [64].

Modified Chang looks for a *consistent* set of prototypes, which is not necessarily a subset of $\mathbf{Z}$. The procedure starts with the whole of $\mathbf{Z}$ as the set of prototypes $\mathbf{V}$. The pair of prototypes of the same class label that are closest to each other is identified and called the "parents." The two prototypes are averaged to get a single replacement of the pair, called the "child." The new set where the two parents are replaced by the child is checked for resubstitution errors. If no errors occur, the merger is accepted and the new set becomes the current $\mathbf{V}$. Otherwise, the merger is rejected and the pair of parents is marked as ineligible. The search continues with the next (eligible) pair of closest prototypes until all remaining parent couples become ineligible.

Using Gradient Descent. Tuning prototype locations by gradient descent is proposed in Refs. [65–67].

Bootstrap Random Methods. Hamamoto et al. [68] propose four bootstrap methods for prototype extraction. A modification of the simplest, and the most successful of the four (according to Ref. [68]) is explained below.

Bootstrap editing is a variant of the random editing method described earlier. Again, we perform T trials, where T is a constant determined in advance. We also have to pick c numbers, v_i, $i = 1, \ldots, c$, where v_i is the number of prototypes from class ω_i that we wish to have in $\mathbf{V}$. At each trial, v elements of $\mathbf{Z}$ are picked at random, v_i from class ω_i. To construct $\mathbf{V}$ from these, each selected element is replaced by the mean of its k nearest neighbors in $\mathbf{Z}$ *from its own class*. The number of neighbors $k > 1$ is a tuning parameter of the algorithm. There is no theoretical reason why the number of prototypes per class should be equal for all classes or proportional to the prior probabilities. We can pick any set of numbers $v_1, \ldots, v_c (v_i \leq N_i)$, or choose them at random too.

Example: Prototype Extraction for the Banana Data. Figure 2.6 shows the regions and the prototypes extracted by the Modified Chang and the bootstrap editing. The bootstrap method was run for $T = 1000$ selections (same as the random editing in the previous example) with 10 prototypes altogether (5 per class), for number of nearest neighbors $k = 5$.

Overlaid again is the training set and the prototypes are marked by "$\times$" and encircled. The training error for Modified Chang is 0 by design and the testing error was found to be 16 percent. In all, 19 prototypes were found. For the bootstrap editing, 10 prototypes were constructed showing training error of 3 percent and testing error of 12 percent.

It should be mentioned that even though the prototype extraction method could theoretically lead to better results, none of the methods described has any theoretical

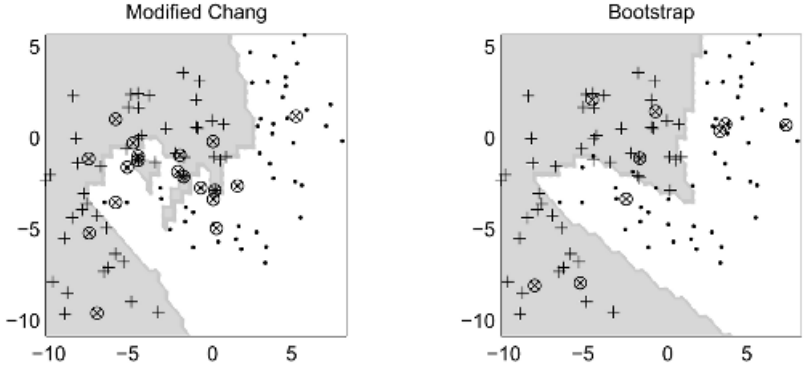


Fig. 2.6 *Training data and classification regions for the banana data set obtained by the 1-nn classifier and two prototype extraction methods. The prototypes retained by the respective method are encircled.*

justification. Both methods (as well as a large gallery of others) rely on heuristics and intuition, which admittedly sometimes work and sometimes do not. So there are no strong guidelines about what to choose: opt for the simple methods first, for example, random and bootstrap editing. Tabu search and LVQ have been found to be good choices from the group of more sophisticated methods [69].

2.3.3 *k*-nn Variants

It is tempting to include in the k-nn rule the *distance* between $\mathbf{x}$ and its k neighbors. Recall the basic nonparametric estimation formula of probability density functions, Eq. (2.23)

$$\hat{p}(\mathbf{x}) = \frac{k}{NV_R} \tag{2.50}$$

where V_R is the volume of a region R in $\Re^n$ containing the k nearest neighbors of $\mathbf{x}$, and N is the total number of observations (cardinality of $\mathbf{Z}$). Denoting by k_i the number of neighbors from class ω_i amongst the k neighbors, and by N_i the total number of elements of $\mathbf{Z}$ from ω_i, the class-conditional pdfs are approximated by

$$\hat{p}(\mathbf{x}|\omega_i) = \frac{k_i}{N_i V_{R_i}} \tag{2.51}$$

Using the approximation

$$\hat{P}(\omega_i) = \frac{N_i}{N} \tag{2.52}$$

for the prior probabilities, the following estimates of the posterior probabilities are obtained

$$\begin{aligned} \hat{P}(\omega_i|\mathbf{x}) &= \frac{k_i}{N_i V_{R_i}} \cdot \frac{N_i}{N} \cdot \frac{1}{p(\mathbf{x})} \\ &= \frac{k_i}{Np(\mathbf{x})V_{R_i}} \end{aligned} \tag{2.53}$$

Assume that R_i is a hypersphere in $\Re^n$ with radius a_i, centered at $\mathbf{x}$. Then the volume of R_i can be expressed using the gamma function Γ

$$V_{R_i}(a_i) = \frac{\pi^{n/2}}{\Gamma\left(\frac{1}{2}n + 1\right)} a_i^n \tag{2.54}$$

which reduces to[8]

$$V_{R_i}(a_i) = V_n(1)\, a_i^n \tag{2.55}$$

where $V_n(1)$ is the volume of a hypersphere of radius 1 in $\Re^n$. Substituting Eq. (2.55) into Eq. (2.53) yields

$$\hat{P}(\omega_i|\mathbf{x}) = \frac{1}{Np(\mathbf{x})V_n(1)} \cdot \frac{k_i}{a_i^n} \tag{2.56}$$

We can ignore the first fraction, which does not depend on i for a given $\mathbf{x}$, and arrive at the following set of simple discriminant functions

$$g_i(\mathbf{x}) = \frac{k_i}{a_i^n}, \qquad i = 1, \ldots, c \tag{2.57}$$

Different interpretations of the above equation give rise to different k-nn variants.

2.3.3.1 Theoretical Model No. 1. Fix $k_i = k$ to be the same for all c classes. Then a_i will be the radius of the hypersphere centered at $\mathbf{x}$ that contains exactly k elements from $\mathbf{Z}$ with class label ω_i. Thus, the radius a_i is the distance between $\mathbf{x}$ and its kth nearest neighbor from ω_i. Since k is the same for all i, the largest $g_i(\mathbf{x})$ will be the one with the smallest distance a_i in it. Therefore, according to Theoretical model no. 1, $\mathbf{x}$ is assigned to the class of the closest kth nearest neighbor.

Example: k-nn: Theoretical Model No. 1. Figure 2.7*a* illustrates this model. The letter data set was used from which we cut the first 50 objects from each of the classes "H" (16), "N" (16), and "O" (18). The data was normalized so that each of the 16 features has mean 0 and variance 1. To plot the data we designed two features so that x is the sum of original features from 1 to 8, and y is the sum of original features from 9 to 16. The figure shows a zoom of the scatterplot of the three classes within the square $[-5, 5]^2$. Class "H" is plotted as dots, "N" as open circles, and "O" as triangles. We use $k = 5$ for each of the three classes. The point to be labeled is the boldface cross at [0, 0]. The three circles correspond to the regions R, each one containing exactly $k = 5$ elements from the respective class, the farthest element (defining a_i) being encircled. The fifth neighbor from class "O" (triangles) is closer to $\mathbf{x}$ than the fifth neighbor from either of the other two classes, therefore $\mathbf{x}$ is labeled as an "O." The label for the same object using $k = 1$ will be "H" (a dot). Notice that for $k = 1$, the model coincides with the nearest neighbor (1-nn) design.

2.3.3.2 Theoretical Model No. 2. Fix k and find $k_1, k_2, \ldots, k_c$, as for the classical k-nn rule. In Eq. (2.57), the radius of the hypersphere centered at $\mathbf{x}$ can be taken

[8] see Calculus books, for example, Ref. [70].

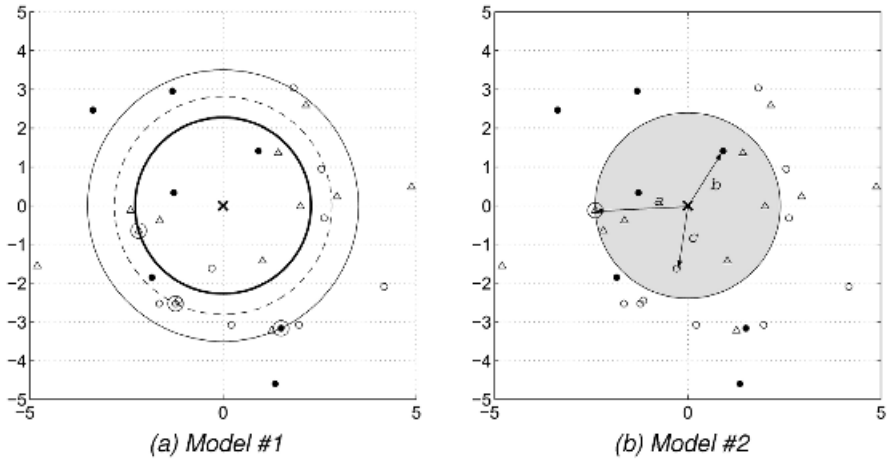


Fig. 2.7 *Illustration of k-nn Theoretical models no. 1 and no. 2.*

to be the distance from $\mathbf{x}$ to its farthest neighbor from ω_i among the k neighbors. Then the ratio k_i/a_i^n determines the class label of $\mathbf{x}$ where k_i is the number of neighbors from ω_i and a_i is the distance to the farthest one of them.

Example: *k*-nn: Theoretical Model No. 2. An illustration of this k-nn model for $k = 9$ is presented in Figure 2.7*b* on the same data set as in Figure 2.7*a*. The shaded circle contains the set of nine nearest neighbors to $\mathbf{x}$, regardless of their class labels. Within this set, there are $k_1 = 2$ "H"s (dots), $k_2 = 1$ "N"s (open circles), and $k_3 = 6$ "O"s (triangles). A (hyper-) sphere with radius a encloses the six triangles, therefore the respective ratio is $g_O(\mathbf{x}) = 6/a^2 = 1.0458$. The other two discriminant functions are respectively $g_H(\mathbf{x}) = 2/b^2 = 0.7080$ and $g_N(\mathbf{x}) = 1/c^2 = 0.3685$. The highest value determines $\mathbf{x}$ class label, in this case the label for [0, 0] is the letter "O." This model also coincides with the baseline 1-nn design for $k = 1$.

2.3.3.3 Theoretical Model No. 3. This is the classical k-nn model where one hypersphere of radius r contains all k nearest neighbors from $\mathbf{Z}$ regardless of the class labels. Then the value of r becomes irrelevant for the classification decision for a given $\mathbf{x}$, and is dropped from Eq. (2.57). The discriminant functions are the number of neighbors k_i.

We can also vary k_i and substitute into Eq. (2.57) the respective radius a_i.

Curiously, all these variants are *asymptotically* Bayes-optimal; that is, for $N \to \infty$, $k \to \infty$ and $k/N \to 0$, Eq. (2.56) produces the true posterior probability, therefore Eq. (2.57) ensures the minimal-error classification. In the finite-sample case, however, there could be better k-nn models. In search of such a model, Dudani proposed a distance-based k-nn [71]. Denote by $\mathbf{Z}^{(\mathbf{x})}$ the subset of $\mathbf{Z}$ containing the k nearest neighbors of $\mathbf{x}$. Dudani's discriminant functions are

$$g_i(\mathbf{x}) = \sum_{\substack{\mathbf{z}_j \in \mathbf{Z}^{(\mathbf{x})}, \\ l(\mathbf{z}_j) = \omega_i}} w_j(\mathbf{x}), \qquad i = 1, \ldots, c \tag{2.58}$$

where $w_j(\mathbf{x})$ are weights calculated by

$$w_j(\mathbf{x}) = \begin{cases} \dfrac{d_k - d_j}{d_k - d_1}, & d_k \neq d_1, \\ 1, & d_k = d_1, \end{cases} \tag{2.59}$$

where d_i is the distance between $\mathbf{x}$ and its ith neighbor. Thus, the nearest neighbor is awarded a weight of 1, and the kth one, the weight of 0 (practically not used in the equation). The author believed that the finite-sample accuracy of the weighted model is superior to that of the unweighted one, and supported this claim with two experimental examples. Dudani's experiments have been criticized for taking all k-nn ties (for the standard method) as errors [72,73]. Bailey and Jain [72] propose three ways of tie-breaking: random, using fewer neighbors, or using more neighbors. They show on the Dudani's experimental set-up that the distance-weighted and the unweighted k-nn are practically indistinguishable. They prove the following theorem

Theorem. (*Bailey and Jain* [72]) *In the infinite sample case* ($N \to \infty$) *the probability of error of the majority k-nearest neighbor rule is minimum among all weighted k-nearest neighbor rules* (*ties are resolved randomly*).

The authors form the difference between the probability of committing an error by any weighted rule T and the majority rule M, given $\mathbf{x}$ and the set of its nearest neighbors $\mathbf{Z}^{(\mathbf{x})}$

$$\Delta_N = P_N(e|\mathbf{x}, \mathbf{Z}^{(\mathbf{x})}, T) - P_N(e|\mathbf{x}, \mathbf{Z}^{(\mathbf{x})}, M) \tag{2.60}$$

and prove that for $N \to \infty$, Δ_N is always nonnegative. Thus, the k-nn rule for a fixed k may not be Bayes-optimal, but it is *asymptotically* the best one among the rules using the same k neighbors. In the finite-sample case, however, it is not clear which of the distance-based models is the best. Macleod et al. [74] state the following (presumably false!) hypothesis

Hypothesis. *The error rate of the unweighted k-nn rule is lower than that of any weighted k-nn rule even when the number of training samples is finite.*

The authors disprove the hypothesis by an example showing analytically that a weighted 2-nn rule ($x \in \Re$, $\Omega = \{\omega_1, \omega_2\}$) gives a lower overall error rate than the corresponding unweighted rule for *any* training set generated from the specified pdfs. This result opens up the door for new distance-based k-nn models. Macleod et al. [74] generalize Dudani's weighting scheme by introducing a parameter α and using the sth nearest neighbor for scaling the distances. The overall formula is the same as Eq. (2.58) but the weights are obtained by

$$w_j(\mathbf{x}) = \begin{cases} \dfrac{d_s - d_j + \alpha(d_s - d_1)}{(1+\alpha)(d_s - d_1)}, & d_s \neq d_1, \\ 1, & d_s = d_1. \end{cases} \tag{2.61}$$

For $\alpha = 0$ and $s = k$, Eq. (2.61) becomes the original formula due to Dudani. The values of these parameters used in the numerical experiments carried out in Ref. [74] were $\alpha \in \{0, 1, 2\}$ and $s \in \{k, 2k, 3k, [N/c], N\}$.

Parthasarathy and Chatterji [75] propose to estimate the posterior probabilities as an average of k estimates of type (2.57). They argue that if the individual estimates are independent and of the same variance θ, the variance of the averaged estimate is k times smaller, θ/k. Neglecting the terms that do not depend on i in the equation proposed in Ref. [75], the discriminant function for class ω_i is

$$g_i(\mathbf{x}) = \sum_{j=1}^{k} \frac{k(\omega_i, j)}{d_j^n} \tag{2.62}$$

where $k(\omega_i, j)$ is the number of neighbors from class ω_i among the nearest j neighbors of $\mathbf{x}$. For example, let $k = 6$, and neighbors 2, 3, and 5 have class label ω_i. Then

$$g_i(\mathbf{x}) = \frac{0}{d_1^n} + \frac{1}{d_2^n} + \frac{2}{d_3^n} + \frac{2}{d_4^n} + \frac{3}{d_5^n} + \frac{3}{d_6^n} \tag{2.63}$$

It is not clear whether the individual estimates $k(\omega_i, j)/d_j^n$ are independent. More likely, they are not, because each set of j neighbors of $\mathbf{x}$, $j = 2, \ldots, k$, already contains the nearest $j - 1$ neighbors used for calculating a previous estimate. The authors assume that the set of discriminant functions (2.62) should be used for classes with equal prior probabilities and suggest another formula where each g_i is multiplied by $\hat{P}(\omega_i)$ for unequal probabilities. However, the estimates of posterior probabilities (2.57) already account for the $P(\omega_i)$, and so does the averaging formula (2.62). Therefore, multiplying the discriminant functions by the prior probabilities only moves the classification boundary towards the more probable class and is theoretically unnecessary as long as our aim is to minimize the error rate.

The list of weighted k-nn rules can be continued further. The problem is that there is practically no guideline as to what to choose. The variety of methods seems to be a hindrance if we are looking for a single method to use. However, in the context of multiple classifier systems, this variety might be extremely useful.

2.4 TREE CLASSIFIERS

Decision tree classifiers are widely used for building classifier ensembles. Three important characteristics of tree classifiers are:

1. If all the objects are distinguishable, that is, there are no identical elements of $\mathbf{Z}$ with different class labels, then we can build a tree classifier with zero resubstitution error. This fact places tree classifiers in the *instable* group: capable of memorizing the training data so that small alterations of the data might lead to a differently structured tree classifier. As we shall see later,

instability can be an advantage rather than a drawback when ensembles of classifiers are considered.

2. Tree classifiers are intuitive because the decision process can be traced as a sequence of simple decisions. Tree structures can capture a knowledge base in a hierarchical arrangement, most pronounced examples of which are botany, zoology, and medical diagnosis.
3. Both quantitative and qualitative features are suitable for building decision tree classifiers.[9] Binary features and features with a small number of categories are especially useful because the decision can be easily branched out. For quantitative features, a point of split has to be found to transform the feature into a categorical one. Thus decision trees do not rely on a concept of distance in the feature space. As discussed earlier, a distance is not easy to formulate when the objects are described by categorical or mixed-type features. This is why decision trees are regarded as *nonmetric methods* for classification [15].

Tree classifiers are usually described in graph terminology. An example is given below.

Example: Terminology of Tree Classifiers. Shown in Figure 2.8*a* is a decision tree for distinguishing between three types of adult bears based upon information found at *http://www.bears-bears.org/*. The three classes are displayed in Figure 2.8*b*.

The features (attributes) are "coloration," "size," and "attack on humans." We chose the features so that the classes be overlapping; that is, there is no 100 percent certainty of correct classification on any of the attributes or any combinations thereof. (Curiously, almost white individuals of the American Black Bear could be found in the northwestern region of North America.)

The first of a sequence of decisions leading finally to a class label is made at the root of the tree by asking a question with a small number of different answers. Depending on the answer, a *branch* is selected and the *child* (*descendent*) node is visited. Another decision is made at this node, and so on, until a *leaf* (a terminal node) is reached. The leaf contains a single class label, which is assigned to the object being classified. The same label may appear at different leaves. If the same number of questions (local decisions) is used to assign the class label of every $\mathbf{x}$, we have a *balanced tree*. Otherwise, the tree is called *imbalanced*. Imbalanced trees reflect the fact that objects that are deep in their classification regions might need shorter decision chains than objects near the classification boundaries.

Usually one feature is used at each nonterminal node (monothetic trees). Subsets of features can be used at a node and the branching decisions can be made by a formula based on these features. The reasons to stick with single features are rather of a

[9] Features are often called *attributes* in the decision tree literature.

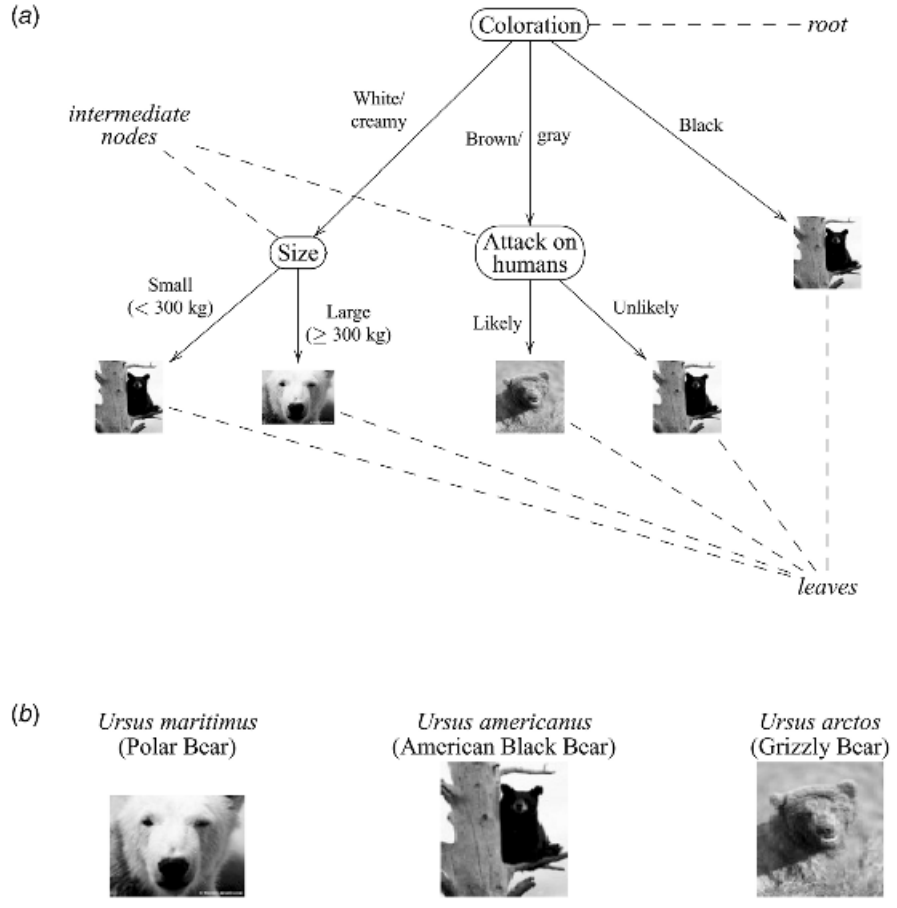


Fig. 2.8 (*a*) *An illustration of a decision tree and related terminology.* (*b*) *Three classes of bear.*

psychological and engineering nature. If more than one feature is involved, the interpretation of the final decision as a chain of *simple* decisions might be difficult or impossible.

To construct a decision tree, we usually start with the root and continue splitting the tree. A "split" means that a separate part of the data set is taken to each child node. That part is subsequently split into smaller parts until a termination criterion is met. A termination criterion could be, for example, that all objects be labeled correctly. Having constructed a perfect tree we have to *prune* it to counter overtraining (called also *postpruning*). Alternatively, we may use some measurable objective function to decide when to stop splitting (called also *prepruning*).

Below we describe a generic tree-growing framework due to Breiman et al. [76], called CART (classification and regression trees). The main questions of this framework and their possible answers are summarized in Ref. [15].

2.4.1 Binary Versus Nonbinary Splits

To automate the tree construction, it is reasonable to choose *binary trees*, that is, each nonterminal node has exactly two children nodes. For any node with multiple answers, there are equivalent representations with binary nodes. For continuous-valued features, the question for a binary split is usually of the form "Is $x \leq x_s$?" where x_s is the node's threshold that is either preset or determined during training. For ordinal categorical features with M successive categories, there are $M - 1$ possible splits into two. For nominal features with M possible values, there are $2^{M-1} - 1$ possible splits (the number of pairs of nonempty subsets).

2.4.2 Selection of the Feature for a Node

The compound objective in designing a tree classifier involves accuracy and simplicity. The construction of the tree splits a given training set hierarchically until either all the objects within the same subregion have the same class label, or the subregion is "*pure enough*."

Consider a c-class problem with $\Omega = \{\omega_1, \ldots, \omega_c\}$. Let P_j be the probability for class ω_j at a certain node t of a decision tree. We can estimate these probabilities as the proportion of points from the respective class within the data set that reached node t. The *impurity* of the distribution of the class labels at t can be measured in different ways.

2.4.2.1 Entropy-Based Measure of Impurity

$$i(t) = -\sum_{j=1}^{c} P_j \log P_j \tag{2.64}$$

where $0 \log 0 = 0$. For a pure region (i.e., only one class label), impurity takes its minimum value, $i(t) = 0$. The most impure situation is when the classes have uniform distribution. In this case, impurity is maximum, $i(t) = \log c$.

2.4.2.2 Gini Impurity

$$i(t) = 1 - \sum_{j=1}^{c} P_j^2 \tag{2.65}$$

Again, for a pure region, $i(t) = 0$. The highest impurity is $i(t) = (c - 1)/c$, in the case of uniform distribution of the class labels. The Gini index can be thought of as the expected classification error incurred if a class label was drawn randomly from the distribution of the labels at t.

2.4.2.3 Misclassification Impurity

$$i(t) = 1 - \max_{j=1}^{c} \{P_j\} \tag{2.66}$$

Misclassification impurity is best related to the classification accuracy. It gives the expected error if the node was replaced by a leaf and the chosen label was the one corresponding to the largest P_j. However, misclassification impurity has the disadvantage of being awkward to handle mathematically. The reason is that the derivative of $i(t)$ is discontinuous and this would be a problem when searching for an optimal threshold value for the node over a continuous-valued feature.

Assume that we split t into two children nodes t_0 and t_1 based on a binary feature X. The gain in splitting t is in the drop of impurity on average, denoted $\Delta i(t)$

$$\begin{aligned}\Delta i(t) &= i(t) - P(X = 0) \times i(t_0) - P(X = 1) \times i(t_1) \\ &= i(t) - P(X = 0) \times i(t_0) - (1 - P(X = 0)) \times i(t_1)\end{aligned} \tag{2.67}$$

where $P(\zeta)$ denotes the probability of event ζ.

If the features being used are binary, then the task of selecting the best feature for node t is easy: try each one in turn and pick the feature with the highest $\Delta i(t)$. However, for features with multiple categories and for continuous-valued features, we have to find first the optimal threshold to split t into left child (t_L) and right child (t_R).

Example: Calculation of Impurity Indices. Figure 2.9 shows the three indices calculated for the generated data set from Figure 1.9. One thousand split points were checked on each axis and the indices Δi were plotted as functions of the split-point. The functions for feature X are projected underneath the data scatterplot, and the functions on feature Y are projected on the left.

All three functions are bimodal on both X and Y, suggesting that there are two good split points on each axis. Indeed, we can choose split points such that either the left or the right subsets are completely pure. Table 2.3 shows the values of Δi for the three indices and the suggested split points. For example, if we use Gini for building the tree, we should prefer the split on X, point $x_s = -0.3458$, because

TABLE 2.3 Maximal Values of Δi and the Respective Split Points for the Three Impurity Indices.

	Entropy		Gini		Misclassification	
	X	Y	X	Y	X	Y
Δi	0.1616	0.1168	0.1127	0.0974	0.0680	0.1230
x_s, y_s	−0.3706	−1.2105	−0.3458	−1.1269	−0.1278	−1.1269

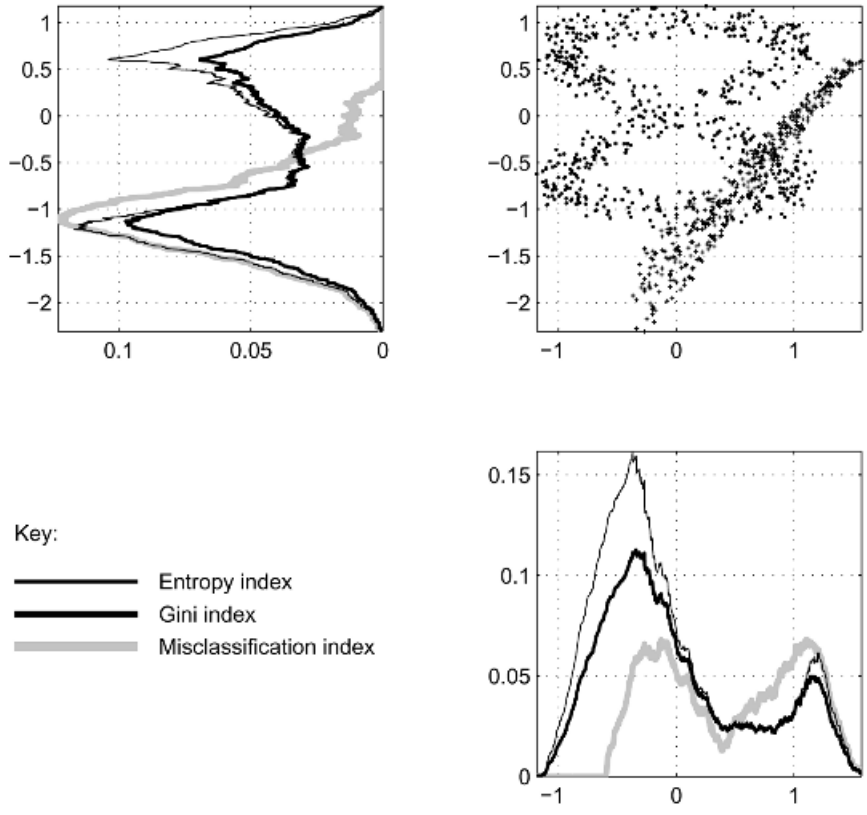


Fig. 2.9 *Three indices of impurity used in growing decision trees: Entropy-based, Gini, and Misclassification indices, calculated for 1000 points of split for each of X and Y, for a generated data set* (*see Figure 1.9*).

splitting on X gives the higher value of Δi. On the other hand, if we used the misclassification index, we should split on Y (higher Δi) at $y_s = -1.1269$. Note that both Entropy and Gini index favor the left peak on X while the Misclassification index favors the right peak. These discrepancies show that different trees will be constructed depending on which index we decide to use.

The Gini index is often used in tree construction [15,29] because it can distinguish between choices for which the misclassification index gives the same value. The choice of impurity index does not seem to be hugely important for the success of the tree classifier [15]; more important are the stopping criteria and the pruning methods.

Note that the choice of the best feature for the current node is a typical example of a *greedy algorithm*. The optimality of the local choices does not guarantee that the constructed tree will be globally optimal, for example, with minimum number of nodes (zero resubstitution error by design).

2.4.3 Stopping Criterion

The tree construction could be carried on until there are no further impure nodes. If the training data contains no objects with the same feature values and different class labels, then a perfect tree can be constructed. However, as discussed earlier, such a perfect result could be due to overtraining, and so the tree classifier would not be of much value. One solution to this problem is to stop the training before reaching pure nodes. How do we decide where to stop? If the splitting is stopped too early, we might leave the classifier undertrained. Below we summarize the options suggested in Ref. [15]:

1. Use a *validation set* cut out from the training set. When the error on the validation set begins to increase, stop splitting.
2. Set a small *impurity-reduction threshold*, β. When the greatest possible reduction of impurity at node t is $\Delta i(t) \leq \beta$, stop splitting and label this node as the majority of the points. This approach does not answer the question "where to stop?" because the stopping is determined by the choice of β, and this choice lies again with the designer.
3. Set a *threshold value for the number of points* at a node. For example, we may decide not to split a node if it contains less than k points or less than l percent of the total number of points in the training set. The argument is that continuing the splitting beyond this point will most probably be overtraining.
4. *Penalize complexity* of the tree by using a criterion such as

$$\text{minimize} \quad \alpha \times \text{size} + \sum_{\text{leaves } t} i(t) \tag{2.68}$$

   where "size" can be the total number of nodes, branches, or leaves of the tree, and α is a positive constant. Again, the problem here is choosing the value of α so that the right balance between complexity and accuracy is achieved.
5. Use *hypothesis testing* to decide whether a further split is beneficial or not. Consider a two-class problem. Assume that there are n data points at node t, n_1 of which have labels ω_1 and n_2 have labels ω_2. Assume that the best feature for the node has been found to be X, and the best split of X is at some point x_s. The left and the right children nodes obtain n_{L} and n_{R} number of points, respectively. Denote by $n_{\text{L}1}$ the number of points from ω_1 in the left child node, and by $n_{\text{L}2}$ the number of points from ω_2 in the left child node. If the distributions of class labels at the children nodes are identical to that at the parent node, then no purity is gained and the split is meaningless. To test the equivalence of the distributions at the children and the parent's node, calculate

$$\chi^2_{\text{L}} = \frac{(n \times n_{\text{L}1} - n_{\text{L}} \times n_1)^2}{n \times n_{\text{L}} \times n_1} + \frac{(n \times n_{\text{L}2} - n_{\text{L}} \times n_2)^2}{n \times n_{\text{L}} \times n_2} \tag{2.69}$$

and

$$\chi^2_{\mathrm{R}} = \frac{(n \times n_{\mathrm{R}1} - n_{\mathrm{R}} \times n_1)^2}{n \times n_{\mathrm{R}} \times n_1} + \frac{(n \times n_{\mathrm{R}2} - n_{\mathrm{R}} \times n_2)^2}{n \times n_{\mathrm{R}} \times n_2} \tag{2.70}$$

We can take the average $\chi^2 = \frac{1}{2}(\chi^2_{\mathrm{L}} + \chi^2_{\mathrm{R}})$, which, after simple algebraic transformations can be expressed as

$$\chi^2 = \frac{1}{2}(\chi^2_{\mathrm{L}} + \chi^2_{\mathrm{R}}) = \frac{n}{2n_{\mathrm{R}}}\chi^2_{\mathrm{L}} = \frac{n}{2n_{\mathrm{L}}}\chi^2_{\mathrm{R}} \tag{2.71}$$

If χ^2 is greater than the tabular value with the specified level of significance and one degree of freedom, then the split is worthwhile. If the calculated value is smaller than the tabulated value, we should stop splitting and should label the node as ω_1 if $n_1 > n_2$, and ω_2 otherwise. The lower the level of significance, the smaller the tree would be because greater differences in the distributions will be required to permit splitting.

For c classes, the degrees of freedom will be $c - 1$, and the χ^2 statistic should be calculated as the average of χ^2_{L} and χ^2_{R} where

$$\chi^2_{\mathrm{L}} = \sum_{i=1}^{c} \frac{(n \times n_{\mathrm{L}i} - n_{\mathrm{L}} \times n_i)^2}{n \times n_{\mathrm{L}} \times n_i} \tag{2.72a}$$

and

$$\chi^2_{\mathrm{R}} = \sum_{i=1}^{c} \frac{(n \times n_{\mathrm{R}i} - n_{\mathrm{R}} \times n_i)^2}{n \times n_{\mathrm{R}} \times n_i} \tag{2.72b}$$

Note that the χ^2 statistic (2.69) or (2.72) can be used for a direct comparison with a threshold set by the user. If the calculated value is greater than the threshold, we should split the node, otherwise we label it as a leaf. Appropriate values of the threshold vary between 0 (full tree) and 10 (heavy prepruning).

Example: Growing Tree Classifiers. Consider the banana data set (see Figure 1.6).[10] To construct the tree we use a recursive procedure of the following general form. Start at the root. At the current node, check whether a split is justified. If yes, call the same procedure for the left subtree and then for the right subtree, then join the two trees. If the split is not justified, then designate the current node as a leaf and label it as the majority of the data points in it.

To find out whether the split is justifiable, we first search for the best split point on every feature. Once such a point is identified, the Gini index of impurity is calculated

[10] The Matlab code for this example is given in Appendix 2A. We borrow the data organization idea from the `maketree` routine from PRTOOLS [19].

TABLE 2.4 A Tabular Description of the Tree Classifier for the Banana Data with Threshold $t = 3$.

Feature (or Class Label)	Split Point	Left Child Node	Right Child Node	
1	1.1740	2	9	
2	−1.5960	3	8	
1	−5.5197	4	5	
2	0	0	0	(leaf)
2	−7.4966	6	7	
2	0	0	0	(leaf)
1	0	0	0	(leaf)
2	0	0	0	(leaf)
1	0	0	0	(leaf)

as in Eq. (2.65) and the feature with the maximal reduction of impurity, $\max \Delta i$, is chosen for that node. To determine the best split point for feature X we first sort the values of X for all the points at the current node and then check Δi for a split point between every two consecutive values of X. The highest Δi signifies the importance of X for the current node and also the best split point.

Finally, the justifiability of the split is assessed by the χ^2 criterion, Eq. (2.69). A split will be permitted only if the calculated χ^2 exceeds the threshold that was set to 3 for this example.

The code `tree_construct` in Appendix 2A outputs a description of the obtained decision tree in the following form. Each row corresponds to a node. If the node is not a leaf, then the first column is the number of the feature used at that node, the second column is the split point, the third and the fourth columns are the numbers of the left and the right children nodes, respectively. If the node

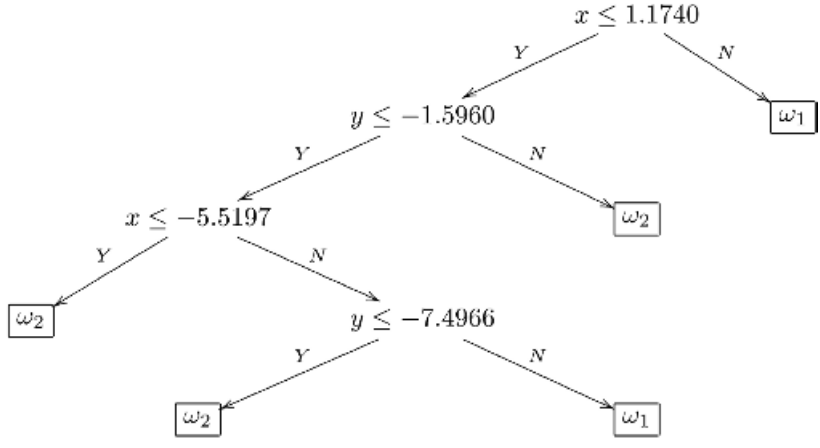


Fig. 2.10 *Graphical representation of the tree classifier for the banana data. A threshold t = 3 was used to preprune the tree. Training error = 5%, testing error = 11%.*

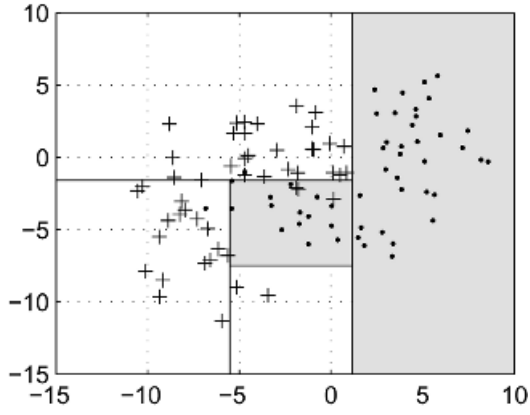


Fig. 2.11 *Classification regions determined by the tree classifier for the banana data.*

is a leaf, then the first column is the class label of the leaf, and columns 2, 3, and 4 contain 0s.

The tree in Table 2.4 was obtained using the code in Appendix 2A for the banana data (Gini index, χ^2 stopping criterion, threshold $t = 3$). Its graphical version is shown in Figure 2.10, and the classification regions are overlaid on the banana data scatterplot in Figure 2.11. It should be noted that exactly the same tree was generated when we used the entropy criterion (2.64). For comparison we grew a full tree with training error 0 percent. The testing error was found to be 13 percent.

One possible way to classify the data after the tree has been built and stored in the format shown in Table 2.4 is given in Appendix 2A, routine `tree_classify`. Using this code, we found that the tree depicted in Figure 2.10 gives 5 percent training error and 11 percent testing error.

2.4.4 Pruning Methods

Sometimes early stopping can be too shortsighted and prevent further beneficial splits. This phenomenon is called the *horizon effect* [15]. To counter the horizon effect, we can grow the full tree and then prune it to a smaller size. The pruning seeks a balance between the increase of the training error and the decrease of the size of the tree. Downsizing the tree will hopefully reduce overtraining. There are different criteria and methods to prune a tree summarized by Esposito et al. [77] as follows.

2.4.4.1 Reduced Error Pruning. The reduced error pruning (REP) method is perceived as the simplest pruning method. It uses an additional training set, called the "pruning set," unseen during the growing stage. Starting at the bottom (leaves) and working our way up to the top (the root), a simple error check is calculated for all nonleaf nodes. We replace the node with a leaf and label it to the majority class,

then calculate the error of the new tree *on the pruning set*. If this error is smaller than the error of the whole tree on the pruning set, we replace the node with the leaf. Otherwise, we keep the subtree. This bottom-up procedure guarantees that the obtained tree is the minimum one that is optimal with respect to the pruning set. Also, REP has low complexity because each internal node is visited just once. A drawback of REP is that it has a tendency towards overpruning [77].

2.4.4.2 Pessimistic Error Pruning. Unlike REP, in the pessimistic error pruning (PEP) method the same data set is used both for growing and pruning the tree. We cannot directly use the error to prune the tree, as done in REP, because on the training data the error rate will be zero or some small $\varepsilon > 0$ due to indiscernible objects in the data (objects with the same feature vectors but different class labels). The approach is top-down, that is, starting from the root and going through all internal nodes that have not been pruned yet. Denote by n the number of data points that reached node t and by $e(t)$, the number of errors if t was replaced by a leaf. Let T_t be the subtree rooted at t, $\mathcal{L}_t$ be the set of leaves of T_t and $e'(T_t)$ the number of errors of T_t with a *complexity correction*

$$e'(T_t) = \sum_{l \in \mathcal{L}_t} e(l) + \frac{|\mathcal{L}_t|}{2} \tag{2.73}$$

The node t is replaced by a leaf if

$$e(t) \leq e'(T_t) + \sqrt{\frac{e'(T_t)[n - e'(T_t)]}{n}} - \frac{1}{2} \tag{2.74}$$

Tree complexity is expressed as the number of leaves of the subtree. Thus a balance is sought between error and complexity but the suitability of the proposed formula is theoretically unjustified. Pessimistic error pruning could lead to underpruning or overpruning.

2.4.4.3 Critical Value Pruning. In the critical value pruning (CVP) method, a critical value is set up as a threshold. The tree is checked in a bottom-up fashion (starting at the intermediate nodes immediately above the leaves and finishing at the root). All the nodes for which the gain in error rate is smaller than the critical value are replaced by leaves. The bottom-up approach overcomes the horizon effect; that is, it ensures that when the gain value at node t is not above the threshold but the values for some of the nodes on the subtree rooted at t are above the threshold, t is not replaced by a leaf. Setting an increasing sequence of critical values, we can design a family of nested trees, the last one being just the root. We can then choose the tree with the minimum error on an independent pruning set. Since the trees were not pruned with respect to the error on this pruning set, there is no guarantee that the most accurate pruned tree derived from the full-grown tree will be found.

2.4.4.4 Cost-Complexity Pruning. The cost-complexity pruning (CCP) method is known as the CART pruning algorithm [76]. In the first step, a parametric family of trees is generated, denoted $\mathcal{T} = \{T_0, \ldots, T_K\}$. Tree T_{i+1} is obtained by pruning tree T_i according to a certain criterion, associated with a parameter value α_{i+1}. For example, α can be the increase in the apparent error rate per leaf, calculated as

$$\alpha = \frac{e(t) - \sum_{l \in \mathcal{L}_t} e(l)}{n(|\mathcal{L}_t| - 1)} \tag{2.75}$$

that is, the additional error if we replaced t by a leaf, divided by the number of leaves we get rid of.[11] If at node t α is smaller than a fixed threshold, then the high complexity and small error reduction of the subtree at node t are not justified and t is replaced by a leaf. T_0 is obtained by pruning the full tree of those branches that have $\alpha = 0$ (no added error), and T_K is the root tree. It can be shown that each T_i has a different α_i such that $\alpha_i < \alpha_{i+1}$. Suppose T_0 is the full tree. To construct T_1, we calculate α for T_0 through Eq. (2.75) for all nonleaf nodes, identify the minimum α, and set this to be our α_1. Then we replace all the nodes whose value is $\alpha = \alpha_1$ by leaves. The so pruned tree is T_1. We continue by identifying the smallest α on T_1, and so on. The second step after constructing the family $\mathcal{T}$ is to find the best tree. This can be done again by using an independent pruning set or by cross-validation [76].

2.4.4.5 Error-Based Pruning. The interesting possibility offered by the error-based pruning (EBP) method is that the nodes can be replaced by children nodes together with the remaining subtree, not only replaced by leaves or kept unchanged. This is called *grafting a branch onto its parent's branch*. Thus the tree can be restructured within the pruning procedure. The criterion that is used for making a decision at node t is based on a statistical estimate of the confidence interval (CI) for the error at t. The upper limit of this CI, U_t, is calculated by treating the set of objects that reached t as a statistical sample and assuming that the number of errors is a binomial random variable. The U_t values are found and subsequently used to calculate the error estimates for the leaves and subtrees. The decision about a node is made based on its expected error, the errors of its children and the error of the largest of the children nodes (taking the largest number of objects).

An extensive experimental study carried out by Esposito et al. [77] found that there are problems that benefit from pruning, others that do not, and a third group where pruning does not lead to significantly different results. Most often pruning was found to be beneficial. The problems where one class is highly dominant, that is, with high $\max_{i=1}^{c} P(\omega_i)$, have been found to benefit from any pruning strategy. There is no unique answer to the question *which is the best pruning method*? Using the above abbreviations, CVP and EBP were found to have a tendency towards overpruning whereas REP has the opposite trend. Finally, it is pointed out that setting aside a pruning subset as a part of the training set does not always

[11] Here n is again the number of data points reaching node t.

pay off. Methods that used the whole training set to grow and then prune the tree were found to be more successful.

The Matlab code `tree_prune` in Appendix 2A implements a pruning method based on the CVP explained above but without going through the two steps. Instead of making a family of trees and selecting the best tree within the family on an independent pruning set, we just use a fixed critical value. The procedure starts with the leaves. Two leaves are merged if χ^2 (2.72) exceeds the fixed threshold (the critical value). We used the code to prune the full tree (25 nodes) for the banana data. We chose the pruning constant to be $t = 3$. The pruned tree consisted of 13 nodes with training accuracy 4 percent and testing accuracy 11 percent.

Example: Tree Pruning for the Banana Data. One hundred pairs of training and testing banana data sets were generated with $\sigma = 1.5$ and 100 data points in each set (50 on each banana). The full tree was grown and pruned varying the threshold from 0 (no pruning) to 10 (heavy pruning). The results were averaged across the 100 pairs of training/testing sets. The training and testing errors are shown in Figure 2.12 and the tree size (number of nodes) is shown in Figure 2.13, all as functions of the threshold t.

The figure suggests that there is an optimal size of the tree for which the testing accuracy is minimum. A pruning threshold around 4 would find this smaller tree. Notice that the tabular value for level of significance 0.05 of the χ^2 test is 3.841.

In terms of accuracy of the final classifier, pruning is perceived to be a better option than early stopping. For small data sets, pruning is inexpensive because grow-

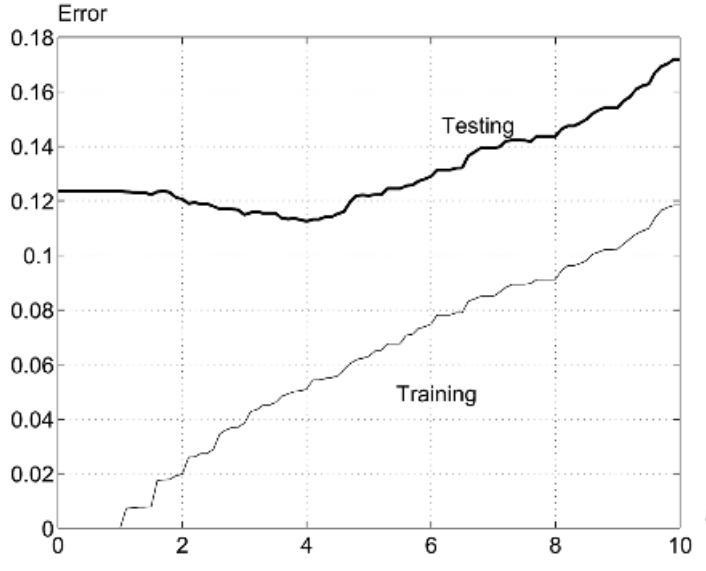


Fig. 2.12 *Training and testing errors of a tree classifier as functions of the pruning threshold (averaged across 100 experiments).*

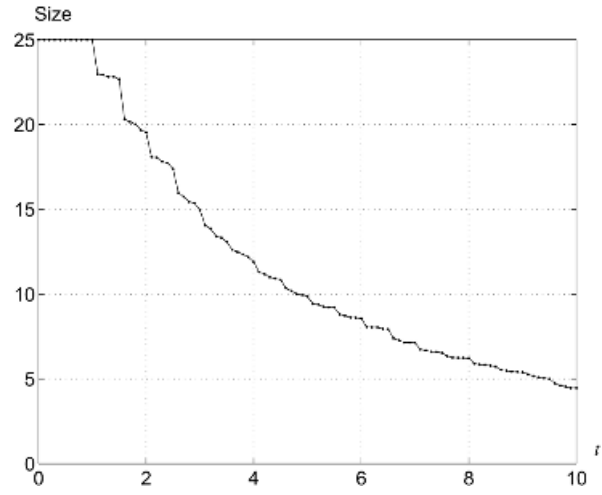


Fig. 2.13 *Size of the tree classifier as a function of the pruning threshold* (*averaged across 100 experiments*).

ing of the full tree is not going to be overly time-consuming. For large data sets, however, the cost of growing the full tree and subsequently pruning it could be prohibitive. Early stopping is recommended for this case.

The methodology for tree construction explained so far is within the CART framework [76]. The two main alternatives for designing trees are the ID3 algorithm and the C4.5 algorithm.

The ID3 algorithm is due to Quinlan [78], a third algorithm from a series of *interactive dichotomizer* algorithms. It is designed for nominal data, so all continuous-valued variables are first discretized and the categories are treated as unordered. The main characteristic of ID3 is that each node has as many children nodes as the number of categories of the (nominal) feature at that node. Since a feature is completely used up on a single node, the tree can only be grown up to maximum n layers, where n is the total number of features. Pruning of the resultant tree can be carried out as discussed above. To select a feature for a node, we should use a modified version of the impurity criteria because the formulas introduced above inherently favor multiple splits over two-way splits without a good reason. Instead of the absolute impurity reduction Δi, we should use a scaled version thereof, called the *gain ratio impurity*. In an M-way split, let P_i be the proportion of objects going into the ith child node. Then

$$\Delta i_M = \frac{\Delta i}{-\sum_{i=1}^{M} P_i \log P_i} \tag{2.76}$$

The main problem with ID3 when continuous-valued variables are concerned is the way these are discretized into categories. There is an inevitable loss of infor-

mation involved in the process. Moreover, a careless formulation of the categories might "kill" an otherwise important variable.

The C4.5 algorithm avoids the discretization problem by using the continuous-valued variables as in CART, and the nominal variables as in ID3. Duda et al. [15] note that C4.5 is the currently most popular tree construction algorithm among the machine learning researchers.

2.5 NEURAL NETWORKS

Artificial neural networks (ANNs or simply NNs) originated from the idea to model mathematically human intellectual abilities by biologically plausible engineering designs. Meant to be massively parallel computational schemes resembling a real brain, NNs evolved to become a valuable classification tool with a significant influence on pattern recognition theory and practice. Neural networks are often used as base classifiers in multiple classifier systems [79]. Similarly to tree classifiers, NNs are instable, that is, small changes in the training data might lead to a large change in the classifier, both in its structure and parameters.

Literature on NNs is excessive and continuously growing. Many publications such as textbooks and monographs [27–29,80–88], paper collections [89], introductory readings [90–92], and so on, discuss NNs at various theoretical and algorithmic depths. Modeling of the human brain, at either morphological or functional level, and trying to understand NNs' cognitive capacity are also important research topics [93–95]. Below we give a brief layout of one of the most popular NN models, the *multilayer perceptron* (MLP).

Consider an n-dimensional pattern recognition problem with c classes. A neural network obtains a feature vector $\mathbf{x} = [x_1, \ldots, x_n]^T \in \Re^n$ at its input, and produces values for the c discriminant functions $g_1(\mathbf{x}), \ldots, g_c(\mathbf{x})$ at its output. Typically NNs are trained to minimize the squared error on a labeled training set $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, $\mathbf{z}_j \in \Re^n$, and $l(\mathbf{z}_j) \in \Omega$

$$E = \frac{1}{2} \sum_{j=1}^{N} \sum_{i=1}^{c} \{g_i(\mathbf{z}_j) - \mathcal{I}(\omega_i, l(\mathbf{z}_j))\}^2 \tag{2.77}$$

where $\mathcal{I}(\omega_i, l(\mathbf{z}_j))$ is an indicator function taking value 1 if the label of $\mathbf{z}_j$ is ω_i, and 0 otherwise. It has been shown that the set of discriminant functions obtained by minimizing Eq. (2.77) approach the posterior probabilities for the classes for data size $N \to \infty$ [96–98]; that is,

$$\lim_{N \to \infty} g_i(\mathbf{x}) = P(\omega_i|\mathbf{x}), \qquad \mathbf{x} \in \Re^n \tag{2.78}$$

This result was brought to light in connection with NNs, but, in fact, it holds for any classifier that can approximate an arbitrary discriminant function with a specified

precision. This *universal approximation* property has been proven for the two important NN models: the *multi-layered perceptron* (MLP) and the *radial basis function* (RBF) networks (for summaries of the literature and proofs refer to Refs. [27] and [29]). Various NN training protocols and algorithms have been developed, and these have been the key to the success of NN classifiers.

2.5.1 Neurons

The processing units in the human brain are neurons of different specialization and functioning. The earliest models of neurons including the model of McCulloch and Pitts [99] and Fukushima's cognitron [100], reprinted in the collection of Ref. [89], were more similar to the biological neuron than later models. For example, they incorporated both activating and veto-type inhibitory inputs. To avoid confusion, artificial neurons are often given other names: "nodes" [101], "units" [27,29], "neurodes" [28]. Simple models will need a large structure for the whole system to work well (e.g., the weightless neural networks [93]), while for more complex models of neurons a few units will suffice. In both cases proper algorithms are needed to train the system (its structure and/or parameters). The basic scheme of a processing node is shown in Figure 2.14.

Let $\mathbf{u} = [u_0, \ldots, u_q]^T \in \Re^{q+1}$ be the input vector to the node and $v \in \Re$ be its output. We call $\mathbf{w} = [w_0, \ldots, w_q]^T \in \Re^{q+1}$ a vector of *synaptic weights*. The processing element implements the function

$$v = \phi(\xi); \qquad \xi = \sum_{i=0}^{q} w_i u_i \tag{2.79}$$

where $\phi : \Re \to \Re$ is the *activation function* and ξ is the *net sum*. Typical choices for ϕ are

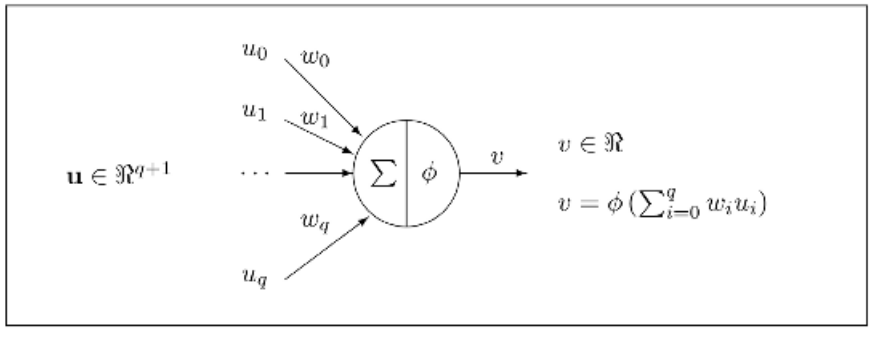


Fig. 2.14 *The NN processing unit.*

- The threshold function

$$\phi(\xi) = \begin{cases} 1, & \text{if } \xi \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.80}$$

- The sigmoid function

$$\phi(\xi) = \frac{1}{1 + \exp(-\xi)} \tag{2.81}$$

- The identity function

$$\phi(\xi) = \xi \tag{2.82}$$

The three activation functions are drawn in Figure 2.15.

The sigmoid is the most widely used one because of the following:

- It can model both linear and threshold functions to a desirable precision; ϕ is almost linear near the origin, whereas for large weights, ϕ is practically the threshold function.
- The sigmoid function is differentiable, which is important for the NN training algorithms. Its derivative on ξ has the simple form $\phi'(\xi) = \phi(\xi)[1 - \phi(\xi)]$.

The weight "$-w_0$" is used as a *bias*, and the corresponding input value u_0 is set to 1. Equation (2.79) can be rewritten as

$$v = \phi[\zeta - (-w_0)] = \phi\left[\sum_{i=1}^{q} w_i u_i - (-w_0)\right] \tag{2.83}$$

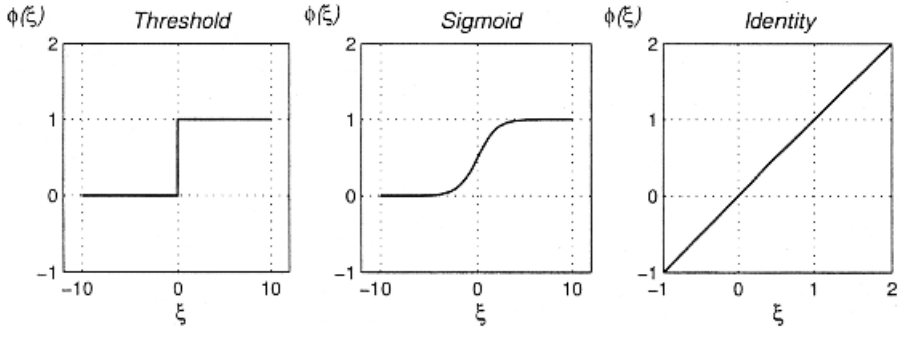


Fig. 2.15 *Threshold, sigmoid, and identity activation functions.*

where ζ is now the weighted sum of the weighted inputs from 1 to q. Geometrically, the equation

$$\sum_{i=1}^{q} w_i u_i - (-w_0) = 0 \tag{2.84}$$

defines a hyperplane in $\Re^q$. A node with a threshold activation function (2.80) responds with value $+1$ to all inputs $[u_1, \ldots, u_q]^T$ on the one side of the hyperplane, and value 0 to all inputs on the other side.

2.5.2 Rosenblatt's Perceptron

Rosenblatt [8] defined the so called *perceptron* and its famous training algorithm. The perceptron is implemented as Eq. (2.79) with a threshold activation function

$$\phi(\xi) = \begin{cases} 1, & \text{if } \xi \geq 0, \\ -1, & \text{otherwise.} \end{cases} \tag{2.85}$$

This one-neuron classifier separates two classes in $\Re^n$ by the linear discriminant function defined by $\xi = 0$. The algorithm starts with random initial weights $\mathbf{w}$ and modifies them as each sample from $\mathbf{Z}$ is subsequently submitted to the input of the perceptron. The modification takes place *only if* the current vector $\mathbf{z}_j$ is misclassified (appears on the wrong side of the hyperplane). The weights are corrected by

$$\mathbf{w} \longleftarrow \mathbf{w} - v\eta\mathbf{z}_j \tag{2.86}$$

where v is the output of the perceptron for $\mathbf{z}_j$ and η is a parameter specifying the *learning rate*. Beside its simplicity, perceptron training has the following interesting properties:

- If the two classes are *linearly separable* in $\Re^n$, the algorithm *always converges in a finite number of steps* to a linear discriminant function that gives no resubstitution errors on $\mathbf{Z}$, for any η. (This is called "the perceptron convergence theorem.")
- If the two classes are not linearly separable in $\Re^n$, the algorithm will loop infinitely through $\mathbf{Z}$ and never converge. Moreover, there is no guarantee that if we terminate the procedure the resultant linear function is the best one found throughout the training.

Appendix 2B contains the code `perceptron` for a perceptron training. To ensure that the program returns an output for both separable and nonseparable classes, a maximum number of 10,000 iterations is set. If this number is reached and there are still misclassified objects in the training set, the program declares

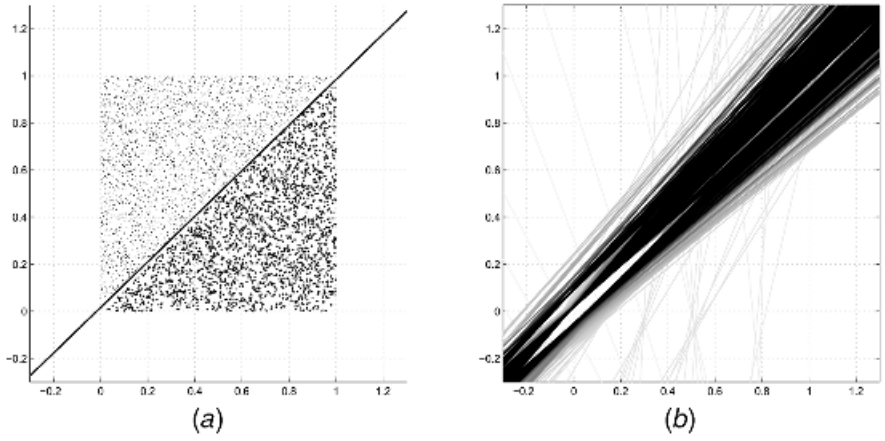


Fig. 2.16 *(a) Uniformly distributed two-class data and the boundary found by the perceptron training algorithm. (b) The "evolution" of the class boundary.*

the classes nonseparable (setting the iteration counter "pass" to 0) and returns the last values of the coefficients $\mathbf{w}$.

Example: Perceptron Training. To illustrate the perceptron training algorithm, a data set of 5000 points was generated, distributed uniformly within the unit square $[0, 1]^2$. Two classes were subsequently labeled, ω_1, containing all points for which $x > y$ and ω_2 with the remaining points. For visualization purposes, the points around the diagonal $x = y$ were removed (all points for which $|x - y| < 0.012$), leaving a small gap between the classes. Figure 2.16*a* shows the data set and the discriminant function found by the perceptron training code in Appendix 2B (the learning rate was $\eta = 0.1$). Eleven passes through $\mathbf{Z}$ were made to achieve separability for the particular random initialization of $\mathbf{w}$.

Figure 2.16*b* depicts the evolution of the class boundary throughout the training. Each time a misclassification occurred and the weight vector changed, we plotted the resultant boundary. Thus even though only eleven passes through $\mathbf{Z}$ were completed, there are as many lines as the total number of misclassifications throughout the training (in this example, 412). Gray lines indicate earlier boundaries whereas black lines were used for the boundaries toward the end of training.

2.5.3 MultiLayer Perceptron

By connecting perceptrons we can design an NN structure called the *multilayer perceptron* (MLP). This is a *feedforward* structure because the output of the input layer and all intermediate layers is submitted only to the higher layer. The generic model of a feedforward NN classifier is shown in Figure 2.17.

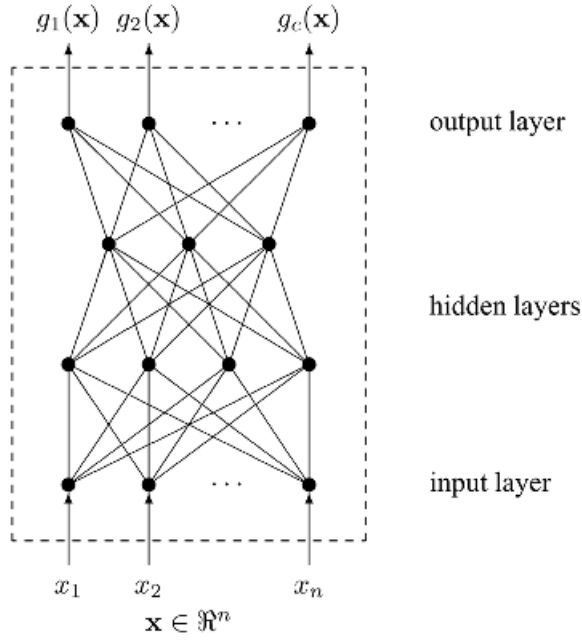


Fig. 2.17 *A generic model of an MLP classifier.*

Here "layer" means a layer of perceptrons. The feature vector $\mathbf{x}$ is submitted to an input layer, and at the output layer there are c discriminant functions $g_1(\mathbf{x}), \ldots, g_c(\mathbf{x})$. The number of hidden layers and the number of perceptrons at each hidden layer are not limited. The most common default conventions are:

- The activation function at the input layer is the identity function (2.82).
- There are no lateral connections between the nodes at the same layer (feed-forward structure).
- Nonadjacent layers are not connected directly.
- All nodes at all hidden layers have the same activation function ϕ.

This model is not as constrained as it might look. In fact, most of the theoretical results in NNs are developed exactly for this model:

- In the late 1980s, based on a simple constructive layout, it was shown that an MLP (as above) with two hidden layers of threshold nodes can approximate *any* classification regions in $\Re^n$ with a specified precision [90,102,103]. Figure 2.18 shows the classification regions that could be formed by an MLP with one, two, and three layers of threshold nodes.

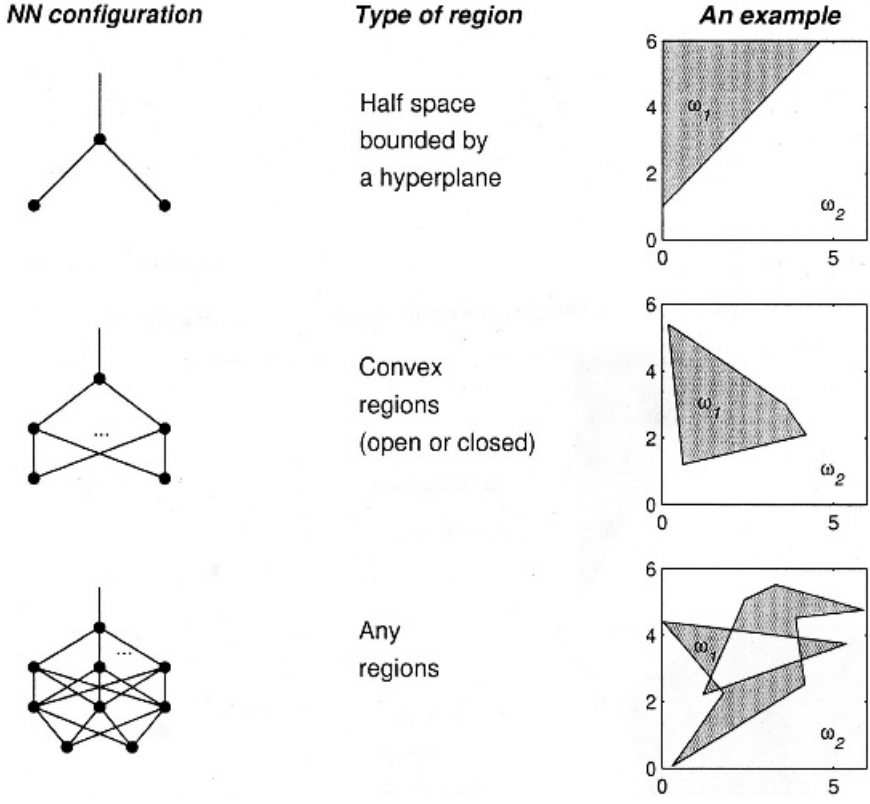


Fig. 2.18 *Possible classification regions for an MLP with one, two, and three layers of threshold nodes.* (*Note that the "NN configuration" column only indicates the number of hidden layers and not the number of nodes needed to produce the regions in column "An example".*)

- It was proven later that even an MLP with a single hidden layer and threshold nodes can approximate *any* function with a specified precision [27,29,104].

The above findings do not tell us *how* to build and train the MLPs and therefore have only theoretical significance. The resurfacing of NN in the 1980s was motivated by the development of the *backpropagation training algorithm*, which provides the means to train a neural network.

2.5.4 Backpropagation Training of MultiLayer Perception

We assume that the structure of the NN is already chosen and fixed (the number of hidden layers and the number of nodes at each hidden layer) and that the activation function is differentiable. The problem is to determine the values of the parameters

(weights) for all nodes. Let θ be a parameter of the NN and $J(\theta)$ be some error function to be minimized. The gradient descent method updates θ by

$$\theta \longleftarrow \theta - \eta \frac{\partial J}{\partial \theta} \tag{2.87}$$

where $\eta > 0$ is the learning rate. An obvious candidate for $J(\theta)$ is the squared error function E of Eq. (2.77). Calculating the derivatives of E on all the weights of the MLP is not straightforward. Consider a node somewhere in the NN with net sum ξ, inputs $u_0, \ldots, u_q$ and weights $w_0, \ldots, w_q$. The derivative of E with respect to w_j is

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial \xi} \frac{\partial \xi}{\partial w_j} = \frac{\partial E}{\partial \xi} u_j \tag{2.88}$$

We call $\delta = \partial E / \partial \xi$ the *error*. Let $\mathbf{x} \in \Re^n$ be a training input with known class label $l(\mathbf{x})$. To calculate the updated weight w_j through Eq. (2.87), we need δ and the inputs u_j of that node for the given $\mathbf{x}$.

We start the training by assigning small random numbers to all weights and selecting a value for the learning rate η. Using the current weights and starting from the input layer, we can calculate subsequently all the u_j in the network (forward propagation). For the δ, however, we proceed backwards, that is, from the output, back to the input (backpropagation). The derivative of E (2.77) with respect to the ith output $g_i(\mathbf{x})$ is

$$\frac{\partial E}{\partial g_i(\mathbf{x})} = g_i(\mathbf{x}) - \mathcal{I}(l(\mathbf{x}), \omega_i) \tag{2.89}$$

Let ξ_i^o be the net sum at output node i and $g_i(\mathbf{x}) = \phi(\xi_i^o)$. Using the chain rule,

$$\delta_i^o = \frac{\partial E}{\partial \xi_i^o} = \frac{\partial E}{\partial g_i(\mathbf{x})} \frac{\partial g_i(\mathbf{x})}{\partial \xi_i^o} = [g_i(\mathbf{x}) - \mathcal{I}(l(\mathbf{x}), \omega_i)] \frac{\partial \phi(\xi_i^o)}{\partial \xi_i^o} \tag{2.90}$$

For the sigmoid activation function ϕ of Eq. (2.81),

$$\delta_i^o = \frac{\partial E}{\partial \xi_i^o} = [g_i(\mathbf{x}) - \mathcal{I}(\mathbf{x}, \omega_i)] g_i(\mathbf{x})[1 - g_i(\mathbf{x})] \tag{2.91}$$

The δ_i^o, $i = 1, \ldots, c$, are the errors of the c nodes at the output layer and are used to update the weights from the last hidden to the output layer. The inputs to every node at the output layer are the outputs of the nodes at the last hidden layer calculated through the forward propagation. Suppose there are M nodes at the last hidden layer with outputs $v_1^h, \ldots, v_M^h$. Each output node will have $v_1^h, \ldots, v_M^h$ plus a bias input $v_0^h = 1$ as its inputs $u_0, \ldots, u_M$ in Eq. (2.88). Having calculated δ_i^o through

Eq. (2.90) or Eq. (2.91), we have now all the necessary components to calculate Eq. (2.88) and update the weights from the last hidden layer to the output layer. Denote by w_{ik}^o the weight of the connection between kth node at the last hidden layer and the ith output. From Eqs. (2.87) and (2.88)

$$w_{ik}^o \leftarrow w_{ik}^o - \eta \delta_i^o v_k^h, \qquad k = 0, \ldots, M, \qquad i = 1, \ldots, c \tag{2.92}$$

Consider now the kth node at the last hidden layer. Let w_{jk}^h denote the weight of the connection coming from the jth node at the previous (hidden or input) layer. We shall refer to the penultimate hidden layer (or input layer) as "$h-1$". Let ξ_k^h be the net sum and $v_k^h = \phi(\xi_k^h)$ be the output of the kth node at the last hidden layer. The error term for the kth node, $\delta_k^h = \partial E / \partial \xi_k^h$, needed for updating all incoming weights from the previous layer, is obtained as

$$\delta_k^h = \frac{\partial E}{\partial \xi_k^h} = \frac{\partial E}{\partial v_k^h} \frac{\partial v_k^h}{\partial \xi_k^h} = \frac{\partial E}{\partial v_k^h} \frac{\partial \phi(\xi_k^h)}{\partial \xi_k^h} \tag{2.93}$$

The first multiplier can be obtained via the chain rule again. Noticing that E depends on v_k^h through the net sums at the output nodes, $\xi_1, \ldots, \xi_c$, and each net sum participates in a separate summation term of E,

$$\frac{\partial E}{\partial v_k^h} = \sum_{i=1}^{c} \frac{\partial E}{\partial \xi_i} \frac{\partial \xi_i}{\partial v_k^h} = \sum_{i=1}^{c} \delta_i^o w_{ki}^o \tag{2.94}$$

The errors $\delta_i^o = \partial E / \partial \xi_i^o$ are already calculated. Using Eq. (2.93), the error at the kth node becomes

$$\delta_k^h = \frac{\partial E}{\partial \xi_k^h} = \left(\sum_{i=1}^{c} \delta_i^o w_{ik}^o \right) \frac{\partial \phi(\xi_k^h)}{\partial \xi_k^h} \tag{2.95}$$

The above equation illustrates the *backpropagation* of the error: to calculate the error term for a node at a certain hidden layer we need the errors calculated already at the higher adjacent layer. For a sigmoid ϕ, Eq. (2.95) becomes

$$\delta_k^h = \frac{\partial E}{\partial \xi_k^h} = \left(\sum_{i=1}^{c} \delta_i^o w_{ik}^o \right) v_k^h (1 - v_k^h) \tag{2.96}$$

To update the weight w_{ik}^h for the connection from the kth node at layer $h-1$ to the ith node at the last hidden layer, we use

$$w_{ik}^h \longleftarrow w_{ik}^h - \eta \delta_i^h v_k^{h-1}, \qquad k = 0, \ldots, S, \quad i = 1, \ldots, M \tag{2.97}$$

where S is the number of nodes at layer $h-1$ and v_k^{h-1}, $k = 1, \ldots, S$ are the outputs at this layer.

Propagating the above formulas backwards through the NN, we calculate the derivatives of E for a single input $\mathbf{x}$. There are two ways to implement the training procedure: batch and on-line. In the *batch* version, the updating (2.87) takes place once after a pass through the whole $\mathbf{Z}$, called an *epoch*. Therefore the derivatives

Backpropagation MLP training

1. Choose an MLP structure: pick the number of hidden layers, the number of nodes at each layer and the activation functions.
2. Initialize the training procedure: pick small random values for all weights (including biases) of the NN. Pick the learning rate $\eta > 0$, the maximal number of epochs T and the error goal $\epsilon > 0$.
3. Set $E = \infty$, the epoch counter $t = 1$ and the object counter $j = 1$.
4. While ($E > \epsilon$ and $t \leq T$) do
   (a) Submit $\mathbf{z}_j$ as the next training example.
   (b) Calculate the output of every node of the NN with the current weights (forward propagation).
   (c) Calculate the error term δ at each node at the output layer by (2.91).
   (d) Calculate recursively all error terms at the nodes of the hidden layers using (2.95) (backward propagation).
   (e) For each hidden and each output node update the weights by

$$w_{new} = w_{old} - \eta \delta u, \tag{2.98}$$

   using the respective δ and u.
   (f) Calculate E using the current weights and Eq. (2.77).
   (g) If $j = N$ (a whole pass through $\mathbf{Z}$ (epoch) is completed), then set $t = t + 1$ and $j = 0$. Else, set $j = j + 1$.
5. End % (While)

Fig. 2.19 *Backpropagation MLP training.*

of E (the update amount $\partial E/\partial w$) are obtained by summing up the derivatives for all $\mathbf{z}_j \in \mathbf{Z}$ for that particular w. Then we apply Eq. (2.87) to every weight of the NN. In the on-line implementation the weights are updated for each $\mathbf{z}_j$ taken as an input to the NN, with the derivatives calculated as above. The *on-line* backpropagation algorithm is explained in Figure 2.19. A simple code for a batch version for an MLP with a single hidden layer (the most frequent choice) is given in Appendix 2B.

The *stochastic* backpropagation version [15] selects examples at random from the training set and updates the weights after the presentation of each selected example. Thus the concept of "epoch" is not relevant for this case.

Example: Backpropagation NN Training. To simplify notations, we shall number consecutively all neurons in the NN, regardless of their layer. Then a weight from i to j will be denoted w_{ij}. Let v_i be the output of neuron i, and δ_i its error. The weight update formula will then take the simple form of

$$w_{ij} \longleftarrow w_{ij} - \eta \times \delta_j v_i \tag{2.99}$$

Figure 2.20 shows an example of an MLP with one hidden layer. The bias nodes, 3 and 7, are taken outside the layers and connected by dashed lines to the respective neurons at the higher level. A bias node has no input and always produces 1 at its output. Since bias nodes can be incorporated in the activation functions, we shall not count them as separate nodes when specifying the NN structure. So the NN in Figure 2.20 has two input, three hidden, and two output neurons. Our NN configuration is conventionally written as 2 : 3 : 2 (input : hidden : output).

Assume that the input neurons 8 and 9 have identity activation functions, neurons 3 and 7 always output 1, and all the remaining neurons use the sigmoid activation function (2.81). We shall demonstrate how the new weights w_{42} and w_{95} are calculated for a single submission of a training example $\mathbf{x}$. Let us start with a random set of weights as shown in Table 2.5*a*. Let $\eta = 0.1$.

Suppose a vector $\mathbf{x} = [2, -1]^T$ from class ω_2 is submitted as the next training example. The forward calculation with the random weights gives the following values of the outputs

$$\begin{array}{llll} v_1 = 0.8488 & v_2 = 0.9267 & & \\ v_3 = 1 & v_4 = 0.7738 & v_5 = 0.8235 & v_6 = 0.9038 \\ v_7 = 1 & v_8 = 2 & v_9 = -1 & \end{array}$$

The target value (class ω_2) is [0, 1]. Using Eq. (2.91), we have

$$\begin{aligned} \delta_1 &= (0.8488 - 0) \times 0.8488 \times (1 - 0.8488) = 0.1089 \\ \delta_2 &= (0.9267 - 1) \times 0.9267 \times (1 - 0.9267) = -0.0050 \end{aligned} \tag{2.100}$$

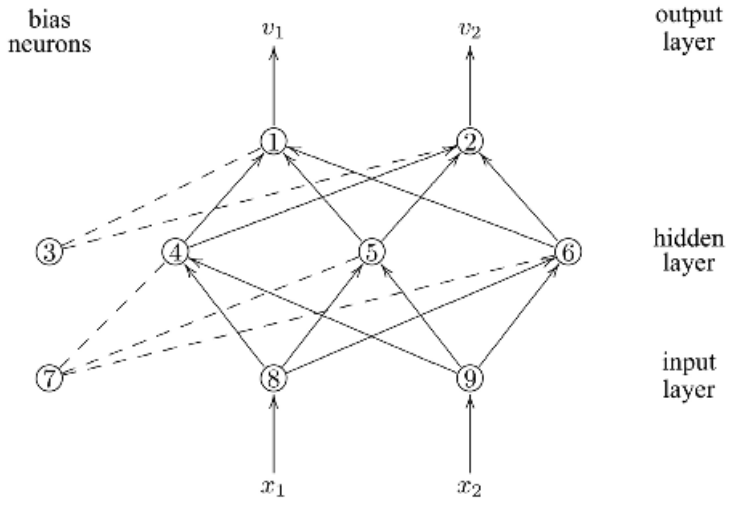


Fig. 2.20 *A 2 : 3 : 2 MLP configuration. Bias nodes are depicted outside the layers and are not counted as separate nodes.*

TABLE 2.5 (a) Random Set of Weights for a 2 : 3 : 2 MLP NN; (b) Updated Weights Through Backpropagation for a Single Training Example.

Neuron		Incoming Weights			
(*a*)	1	$w_{31} = 0.4300$	$w_{41} = 0.0500$	$w_{51} = 0.7000$	$w_{61} = 0.7500$
	2	$w_{32} = 0.6300$	$w_{42} = 0.5700$	$w_{52} = 0.9600$	$w_{62} = 0.7400$
	4	$w_{74} = 0.5500$	$w_{84} = 0.8200$	$w_{94} = 0.9600$	
	5	$w_{75} = 0.2600$	$w_{85} = 0.6700$	$w_{95} = 0.0600$	
	6	$w_{76} = 0.6000$	$w_{86} = 1.0000$	$w_{96} = 0.3600$	
(*b*)	1	$w_{31} = 0.4191$	$w_{41} = 0.0416$	$w_{51} = 0.6910$	$w_{61} = 0.7402$
	2	$w_{32} = 0.6305$	$w_{42} = 0.5704$	$w_{52} = 0.9604$	$w_{62} = 0.7404$
	4	$w_{74} = 0.5500$	$w_{84} = 0.8199$	$w_{94} = 0.9600$	
	5	$w_{75} = 0.2590$	$w_{85} = 0.6679$	$w_{95} = 0.0610$	
	6	$w_{76} = 0.5993$	$w_{86} = 0.9986$	$w_{96} = 0.3607$	

Propagating the error to the hidden layer as in Eq. (2.96), we calculate

$$\begin{aligned}\delta_4 &= (\delta_1 \times w_{41} + \delta_2 \times w_{42}) \times v_4 \times (1 - v_4) \\ &= (0.1089 \times 0.05 - 0.0050 \times 0.57) \times 0.7738 \times (1 - 0.7738) \\ &\approx 0.0005 \end{aligned} \tag{2.101}$$

In the same way we obtain $\delta_5 = 0.0104$ and $\delta_6 = 0.0068$. We can now calculate the new values of all the weights through Eq. (2.98). For example,

$$w_{42} = w_{42} - \eta \times \delta_2 \times v_4$$

$$= 0.57 - 0.1 \times (-0.0050) \times 0.7738 \tag{2.102}$$

$$= 0.5704 \tag{2.103}$$

For input-to-hidden layer weights we use again Eq. (2.98); for example,

$$w_{95} = w_{95} - \eta \times \delta_5 \times v_9 = w_{95} - \eta \times \delta_5 \times x_2 \tag{2.104}$$

$$= 0.06 - 0.1 \times 0.0104 \times (-1) = 0.0610 \tag{2.105}$$

The new values of the weights, updated after the presentation of the single example $\mathbf{x}$ are given in Table 2.5*b*.

Example: 2 : 3 : 2 MLP for the Banana Data. This example illustrates the backpropagation training for the banana data set. We chose the NN configuration depicted in Figure 2.20 and used in the example above. The training protocol was as follows:

- version of backpropagation: batch;
- maximum number of epochs: $T = 1000$;
- learning rate: $\eta = 0.1$;
- error goal: 0;
- initialization of the weights and biases: random numbers in the interval [0, 1].

The function `backprop` given in Appendix 2B was used to train the NN. We measured the squared error at each epoch (the criterion function) and also the appar-

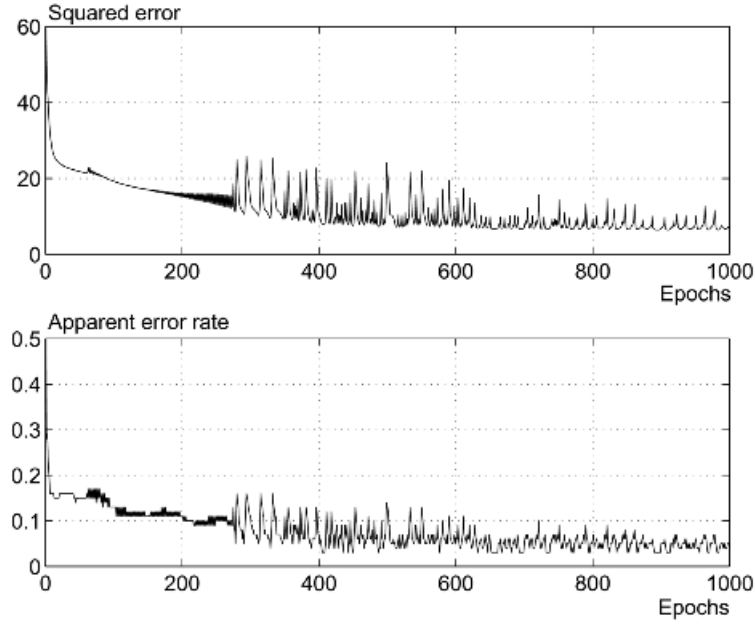


Fig. 2.21 *Squared error and the apparent error rate versus the number of epochs for the backpropagation training of a 2 : 3 : 2 MLP on the banana data.*

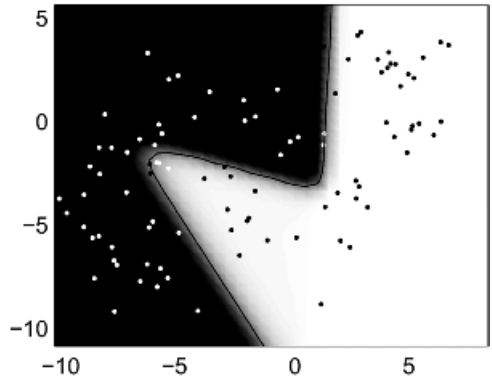


Fig. 2.22 *The classification regions for the banana data produced by the trained 2 : 3 : 2 MLP.*

ent error rate. Although generally corresponding, the two are not identical. Figure 2.21 plots the two errors calculated on the training set versus the iteration count.

As expected, both errors decline, and towards the end of the training process oscillate along some small residual value. Since the squared error was not brought down to the target value, 1000 iterations (epochs) were carried out. Figure 2.22 shows the classification regions, shaded with different gray intensity depending on the value of the NN output. The training error at the 1000th epoch was 4 percent and the testing error was 9 percent.

There are a number of interesting modifications of backpropagation training aiming at higher effectiveness (e.g., faster convergence, stability, and so on) [28].

APPENDIX 2A MATLAB CODE FOR TREE CLASSIFIERS

The below collection of functions produces a tree classifier for a given data set Z with numerical labels in "labZ." The labels should be consecutive integers starting with 1. The number of classes c is also needed as an input parameter. The tree is stored in the output array T, as explained in the text. The function `tree_construct` uses a threshold which was assigned to 3 in the calculations for the examples given in the text.

```
function T=tree_construct(Z,labZ,c,chi2_threshold);
if size(Z,1)>1,
    if min(labZ)==max(labZ),
        % one class -->
        % do not split and make a leaf labeled labZ(1)
        T=[labZ(1),0,0,0];
    else
        [j,s]=tree_select_feature(Z,labZ,c);
```

```
        chi2=tree_chi2(Z,labZ,c,j,s);
        if (chi2>chi2_threshold)&(s~=-999),
            left=Z(:,j)<=s;
            % accept the split
            Tl=tree_construct(Z(left,:),...
                labZ(left),c,chi2_threshold);
            % left subtree
            Tr=tree_construct(Z(~left,:),...
                labZ(~left),c,chi2_threshold);
            % right subtree
            % merge the two trees
            Tl(:,[3 4])=Tl(:,[3 4])+(Tl(:,[3 4])>0)*1;
            Tr(:,[3 4])=Tr(:,[3 4])+...
                        (Tr(:,[3 4])>0)*(size(Tl,1)+1);
            T=[j,s,2,size(Tl,1)+2;Tl;Tr];
        else
            % do not split and make a leaf
            % labeled as the majority
            for i=1:c,
                count(i)=sum(labZ==i);
            end;
            [dummy,majority_label]=max(count);
            T=[majority_label,0,0,0];
        end;
    end;
else
    T=[labZ(1),0,0,0]; % there is only one point in Z
end;

function [maxj,maxs]=tree_select_feature(Z,labZ,c);
[n,m]=size(Z);
i_G=Gini(labZ);
% Gini index of impurity at the parent node
for j=1:m,
    if min(Z(:,j))==max(Z(:,j)),
        % the feature has only one value
        D=0;
        s=-999; % missing value code
    else
        [Zsrt,iZsrt]=sort(Z(:,j)); % sort j-th feature
        for i=1:n-1, % check the n-1 split points
            sp=(Zsrt(i)+Zsrt(i+1))/2;
            left=Z(:,j)<=sp;
            if (sum(left)>0)&(sum(left)<n),
                % there are points in both children nodes
                i_GL=Gini(labZ(left));
                i_GR=Gini(labZ(~left));
                Delta_i(i)=i_G-mean(left)*i_GL ...
                                  -mean(~left)*i_GR;
            else
               % one node is empty
               Delta_i(i)=0;
            end;
        end;
        [D(j),index_s]=max(Delta_i);
```

```
            % maximum of Delta_i for the jth feature
        s(j)=(Zsrt(index_s)+Zsrt(index_s+1))/2;
    end;
end;
[maxD,maxj]=max(D); % maxj is the best feature
maxs=s(maxj); % maxs is the corresponding split point

function chi2=tree_chi2(Z,labZ,c,j,s);
n=size(Z,1);
chi2=0;
left=Z(:,j)<=s;
n_L=sum(left);
for i=1:c,
    n_i=sum(labZ==i);
    n_iL=sum(labZ(left)==i);
    if (n_i>0)&(n_L>0)
        chi2=chi2+(n*n_iL-n_i*n_L)^2/(2*n_i*(n_L)*(n-n_L));
    end;
end;

function i_G=Gini(labZ);
for i=1:max(labZ),
    P(i)=mean(labZ==i);
end;
i_G=1-P*P';

function [guessed_lab,e]=tree_classify(T,Z,labZ);
% T is a tree constructed by tree_construct
% e is the error rate of the tree classifier for
% the data set Z with labels in labZ
for i=1:size(Z,1),
    ind=1;
    leaf=0;
    while leaf==0,
        if T(ind,3)==0,% leaf is found
            guessed_lab(i)=T(ind,1);
            leaf=1;
        else
            if a(i,T(ind,1))<=T(ind,2),
                ind=T(ind,3); % left child
            else
                ind=T(ind,4); % right child
            end;
        end;
    end;
end;
e=1-mean(labZ==guessed_lab');

function Tpr=tree_prune(T,Z,labZ,c,chi2_threshold);

% Find the distributions of points at each node of T
Tdistr=zeros(size(T,1),c); % initialize a distribution array
for i=1:size(Z,1), % for each data point in Z, check which nodes
                   % it goes through, and increment the respective
                   % distributions
```

```
    ind=1;
    leaf=0;
    while leaf==0,
        Tdistr(ind,labZ(i))=Tdistr(ind,labZ(i))+1; % increment counter
                    % for class lsbZ(i) at the visited node i
        if T(ind,3)==0, % leaf is found
            leaf=1;
        else
            if Z(i,T(ind,1))<=T(ind,2),
               ind=T(ind,3); % left
            else
               ind=T(ind,4); % right
            end;
        end;
    end;
end;

prune=1; % indicator of changes of T,
while prune==1,
    prune=0;
    for i=1:size(T,1),
        if T(i,3)>0, % not a leaf
            if (T(T(i,3),3)==0)&(T(T(i,4),3)==0),
                % if both children are leaves, consider pruning
                n=sum(Tdistr(i,:));
                chi2=0;
                n_L=sum(Tdistr(T(i,3),:));
                for k=1:c,
                    if Tdistr(i,k)>0,
                        chi2=chi2+(n*Tdistr(T(i,3),k)-...
                        Tdistr(i,k)*n_L)^2 /(n*Tdistr(i,k)*n_L);
                    end;
                end;
                n_R=sum(Tdistr(T(i,4),:));
                chi2=chi2*(n_L+n_R)/(2*n_R);
                if chi2<chi2_threshold % merge
                    prune=1; % set the indicator
                    [dummy,T(i,1)]=max(Tdistr(i,:));
                    % label the i-th node
                    T(T(i,3),:)=zeros(1,4);
                    % delete the children nodes
                    T(T(i,4),:)=zeros(1,4);
                    T(i,2)=0;T(i,3)=0;T(i,4)=0;
                end;
            end;
        end;
    end;
end;

% Pack T
Tpr=T;
for i=1:size(T,1),
    if T(i,1)==0,
        for k=1:size(T,1),
            if T(k,3)>i;Tpr(k,3)=Tpr(k,3)-1;end;
```

```
            if T(k,4)>i;Tpr(k,4)=Tpr(k,4)-1;end;
        end;
    end;
end;
Tpr=Tpr(T(:,1)>0,:);
```

APPENDIX 2B MATLAB CODE FOR NEURAL NETWORK CLASSIFIERS

```
function [w,pass]=perceptron(Z,labZ,eta);
ier=1; % initialize a misclassification indicator
pass=0; % initialize iteration counter
w=rand(1,size(Z,2)+1); % initialize weights + threshold
while ier==1, % misclassifications occurred in the
              % last pass through Z
    ier=0;
    for j=1:size(Z,1), % a pass through Z
        y=sign([Z(j,:) 1]*w'); % perceptron output
        if y*sign(labZ(j)-1.5)<1
             % misclassification
             ier=1; % switch the indicator
             w=w-eta*y*[Z(j,:) 1]; % update weight vector
        end;
    end;
    pass=pass+1;
    if pass=10000,
        ier=0;
        pass=0;
    end;
end;

function [W1,B1,W2,B2,estored,miscl]=...
backprop(Z,labZ,M,T,epsilon,eta);
% Backpropagation training of an MLP,
% with a single hidden layer with M nodes in it
%
% W1 and W2 are arrays with weights: input-hidden
% and hidden-output, respectively
%======= INITIALIZATION ========
[n,m]=size(Z);c=max(labZ);
bin_labZ=repmat(labZ,1,c)==repmat(1:c,n,1); % use labels
            % to form c-component binary target vectors
W1=rand(M,m); % weights input-hidden
B1=rand(M,1); % biases input-hidden
W2=rand(c,M); % weights hidden-output
B2=rand(c,1); % biases hidden-output
E=inf; % criterion value
t=1; % iteration counter
%======== CALCULATION==========
estored=[];miscl=[];
while (E>epsilon) & (t<=T),
    oh=1./(1+exp(-[W1 B1]*[Z ones(n,1)]'));
```

```
        % outputs of the hidden layer
    o=1./(1+exp(-[W2 B2]*[oh; ones(1,n)]));
        % outputs of the output layer
    E=sum(sum((o'-bin_labZ).^2));
    delta_o=(o-bin_labZ').*o.*(1-o);
    delta_h=[(delta_o'*W2).*oh'.*(1-oh')]';
    for i=1:c, % update W2 and B2
        for j=1:M,
             W2(i,j)=W2(i,j)-eta*delta_o(i,:)*[oh(j,:)]';
        end;
        B2(i)=B2(i)-eta*delta_o(i,:)*ones(n,1);
    end;
    for i=1:M, % update W1 and B1
        for j=1:m,
             W1(i,j)=W1(i,j)-eta*delta_h(i,:)*Z(:,j);
        end;
        B1(i)=B1(i)-eta*delta_h(i,:)*ones(n,1);
    end;
    t=t+1;
    estored=[estored;E]; % store the MLP squared error
    [dummy,guessed_labels]=max(o);
    miscl=[miscl;1-mean(guessed_labels'==labZ)];
        % store the classification error
end;
```

3

Multiple Classifier Systems

3.1 PHILOSOPHY

By combining classifiers we are aiming at a more accurate classification decision at the expense of increased complexity. The question is whether a combination of classifiers is justified. Are we not bringing the fundamental pursuit in pattern recognition to a different level? Ho expresses this concern in her critical review article "Multiple classifier combination: Lessons and the next steps" [105]:

> Instead of looking for the best set of features and the best classifier, now we look for the best set of classifiers and then the best combination method. One can imagine that very soon we will be looking for the best set of combination methods and then the best way to use them all. If we do not take the chance to review the fundamental problems arising from this challenge, we are bound to be driven into such an infinite recurrence, dragging along more and more complicated combination schemes and theories and gradually losing sight of the original problem.

The warning here is that we should be able to make the best use of the tools and methodologies that we have at present, before setting off for new complicated designs. Combining classifiers appears as a natural step forward when a critical mass of knowledge of single classifier models has been accumulated. Although there are many unanswered questions about matching classifiers to real-life problems, combining classifiers is rapidly growing and enjoying a lot of attention from pattern recognition and machine learning communities.

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

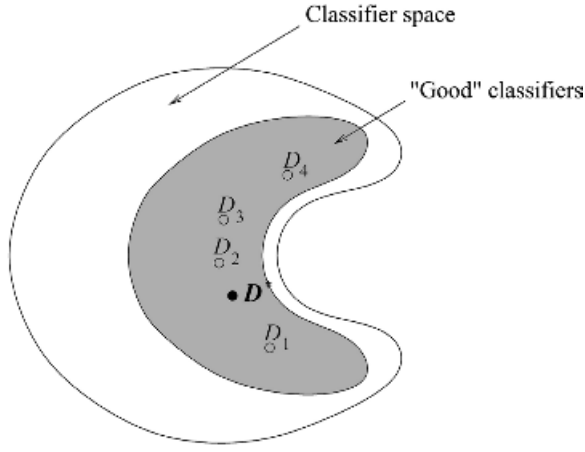


Fig. 3.1 *The statistical reason for combining classifiers. D^* is the best classifier for the problem, the outer curve shows the space of all classifiers; the shaded area is the space of classifiers with good performances on the data set.*

Dietterich [106] suggests three types of reasons why a classifier ensemble might be better than a single classifier.

3.1.1 Statistical

Suppose we have a labeled data set $\mathbf{Z}$ and a number of different classifiers with a good performance on $\mathbf{Z}$. We can pick a single classifier as the solution, running onto the risk of making a bad choice for the problem. For example, suppose that we run the 1-nn classifier or a decision tree classifier for L different subsets of features thereby obtaining L classifiers with zero resubstitution error. Although these classifiers are indistinguishable with respect to their (resubstitution) training error, they may have different generalization performances. Instead of picking just one classifier, a safer option would be to use them all and "average" their outputs. The new classifier might not be better than the single best classifier but will diminish or eliminate the risk of picking an inadequate single classifier.

Dietterich gives a graphical illustration of this argument as shown in Figure 3.1.[12] The outer circle denotes the space of all classifiers. The shaded inner region contains all classifiers with good performances on the training data. The best classifier for the problem (supposedly with a good performance on the training data too) is denoted by D^*. The hope is that some form of aggregating of the L classifiers will bring the resultant classifier closer to D^* than a classifier randomly chosen from the classifier space would be.

[12] The appearance and notations in the figures inspired by Ref. [106] are changed so as to be consistent with the rest of the book.

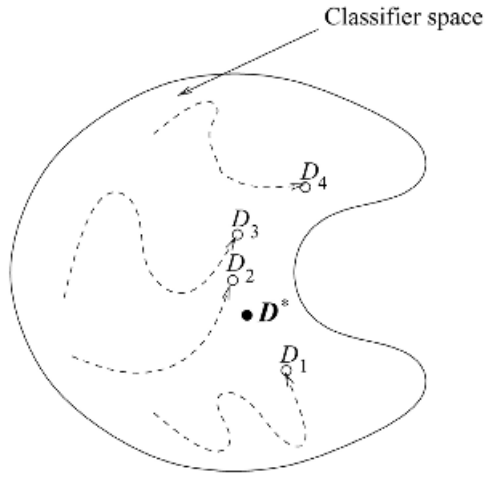


Fig. 3.2 *The computational reason for combining classifiers. D^* is the best classifier for the problem, the closed space shows the space of all classifiers, the dashed lines are the hypothetical trajectories for the classifiers during training.*

3.1.2 Computational

Some training algorithms perform hill-climbing or random search, which may lead to different local optima. Figure 3.2 depicts this situation. We assume that the training process of each individual classifier starts somewhere in the space of possible classifiers and ends closer to the optimal classifier D^*. Some form of aggregating may lead to a classifier that is a better approximation to D^* than any single D_i.

3.1.3 Representational

It is possible that the classifier space considered for the problem does not contain the optimal classifier. For example, the optimal classifier for the banana data discussed earlier is nonlinear. If we restrict the space of possible classifiers to linear classifiers only, then the optimal classifier for the problem will not belong in this space. However, an ensemble of linear classifiers can approximate any decision boundary with any predefined accuracy. If the classifier space is defined differently, D^* may be an element of it. In this case, the argument is that training an ensemble to achieve a certain high accuracy is more straightforward than training directly a classifier of high complexity. For example, a single neural network can be trained for the problem instead of looking at a combination of simple classifiers. When there are many parameters (weights) to be tuned, a local extremum of the error function is likely to be found. An ensemble of simple classifiers might be a better option for such problems. Figure 3.3 illustrates the case where the optimal classifier D^* is outside the chosen space of classifiers.

Note that an improvement on the single best classifier or on the group's average performance, for the general case, is not guaranteed. What is exposed here are only

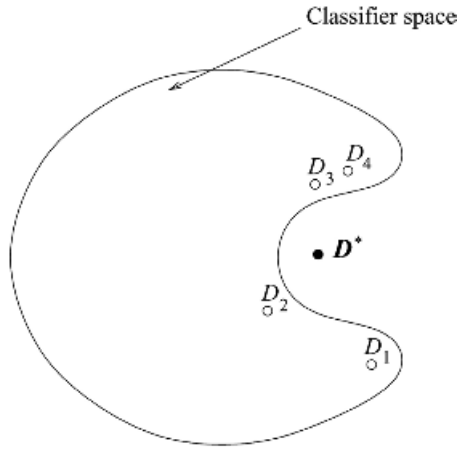


Fig. 3.3 *The representational reason for combining classifiers. D^* is the best classifier for the problem; the closed shape shows the chosen space of classifiers.*

"clever heuristics." However, the experimental work published so far and the theories developed for a number of special cases demonstrate the success of classifier combination methods.

3.2 TERMINOLOGIES AND TAXONOMIES

The series of annual International Workshops on Multiple Classifier Systems (MCS), held since 2000, has played a pivotal role in organizing, systematizing, and developing further the knowledge in the area of combining classifiers [107–110]. We still do not have an agreed upon structure or a taxonomy of the whole field, although a silhouette of a structure is slowly crystallizing among the numerous attempts. Providing yet another taxonomy is not the intention of this chapter. We will rather look at several popular ways to summarize the work in the field in the hope that a structure will be found in the future.

Before the series of MCS workshops, classifier combination went through parallel routes within pattern recognition and machine learning, and perhaps in other areas such as data fusion. This brought various terms for the same notion. We note that some notions are not absolutely identical but bear the specific flavor of their area of origin. For example

$$\text{classifier} = \text{hypothesis} = \text{learner} = \text{expert}$$

and

$$\text{example} = \text{instance} = \text{case} = \text{data point} = \text{object}$$

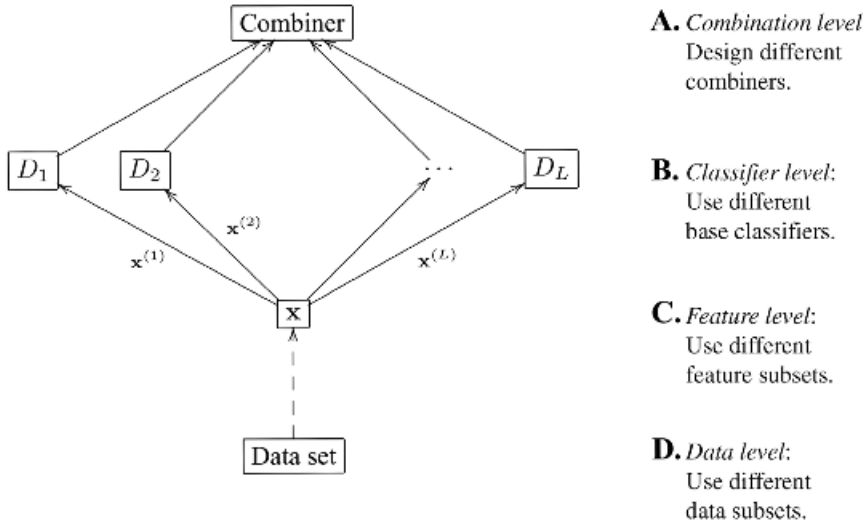


Fig. 3.4 *Approaches to building classifier ensembles.*

A similar variety of terminology can be observed in the toolbox of methods for classifier combination.

A starting point for grouping ensemble methods can be sought in the ways of building the ensemble. The diagram in Figure 3.4 illustrates four approaches aiming at building ensembles of diverse classifiers.

This book is mainly focused on Approach A. Chapters 4, 5, and 6 contain details on different ways of combining the classifier decisions. The base classifiers (Approach B) can be any of the models discussed in Chapter 2 along with classifiers not discussed in this book. Many ensemble paradigms employ the same classification model, for example, a decision tree or a neural network, but there is no evidence that this strategy is better than using different models. The design of the base classifiers for the ensemble is partly specified within the *bagging and boosting models* (Chapter 7) while designing the combiner is not coupled with a specific base classifier. At feature level (Approach C) different feature subsets can be used for the classifiers. This topic is included in Chapter 8. Finally, the data sets can be modified so that each classifier in the ensemble is trained on its own data set (Approach D). This approach has proven to be extremely successful owing to the bagging and boosting methods described in Chapter 7.

Although many of the existing streams in classifier combination are captured in the four-approaches classification, there are many more that are left outside. For example, a remarkably successful ensemble building heuristic is manipulating the output labels by using *error correcting codes* (ECOC) (Chapter 8). Other topics of interest include clustering ensembles (Chapter 8) and diversity in classifier ensembles (Chapter 10). Developing a general theory, as impossible as it sounds,

is an attractive goal in classifier combination. Chapter 9 provides a compilation of some published theoretical results. Certainly many more methods and paradigms have been developed, especially those tailored to real-life applications. Such methods are even more interesting than the general ones, but at the same time very hard to summarize in the same text.

Below we sketch some of the views in the literature on grouping classifier methods.

3.2.1 Fusion and Selection

It is accepted now that there are two main strategies in combining classifiers: fusion and selection. In classifier fusion, each ensemble member is supposed to have knowledge of the whole feature space. In classifier selection, each ensemble member is supposed to know well a part of the feature space and be responsible for objects in this part. Therefore in the fusion approach, we apply combiners such as the average and majority vote whereas in the selection approach we usually select one classifier to label the input $\mathbf{x}$. There are combination schemes lying between the two "pure" strategies. Such a scheme, for example, is taking the average of the outputs with coefficients that depend on the input $\mathbf{x}$. Thus the local (with respect to $\mathbf{x}$) competence of the classifiers is measured by the weights. Then more than one classifier is responsible for $\mathbf{x}$ and the outputs of all responsible classifiers are fused. The mixture of experts architecture (Chapter 6) is an example of a scheme between selection and fusion.

The fusion–selection pair has the following synonyms in the literature

fusion–selection
competitive classifiers–cooperative classifiers
ensemble approach–modular approach [79]
multiple topology–hybrid topology [111]

Classifier selection has not attracted as much attention as classifier fusion. This might change in the future as classifier selection is probably the better of the two strategies, if trained well. Cascade classifiers also seem to be relatively neglected although they could be of primary importance for real-life applications.[13]

3.2.2 Decision Optimization and Coverage Optimization

Decision optimization refers to methods to choose and optimize the combiner for a fixed ensemble of base classifiers (Approach A in Figure 3.4) [105]. Coverage optimization refers to methods for creating diverse base classifiers assuming a

[13]A cascade classifier is an important version of classifier selection where only one classifier is active at a time and the rest of the ensemble are dormant. When an $\mathbf{x}$ is submitted, the first classifier tries to make a decision. If the classifier is "certain" in its decision, $\mathbf{x}$ is labeled and the procedure stops. If not, $\mathbf{x}$ is passed on to the second classifier and is processed in the same way.

fixed combiner (Approaches B, C, and D). The following correspondences can be found in the literature

decision optimization–coverage optimization [105]
nongenerative ensembles–generative ensembles [112]

Note that the decision–coverage grouping looks at the ensemble methods from a different perspective than the fusion-selection grouping. The fusion–selection grouping can be thought of as a subdivision of the decision optimization group. However, some methods from the fusion–selection division come with their own coverage optimization plan. For example, the mixture of experts model trains the classifiers and the combiner simultaneously. Thus it cannot be classed as either a decision optimization or a coverage optimization method.

3.2.3 Trainable and Nontrainable Ensembles

Some combiners do not need training after the classifiers in the ensemble have been trained individually. An example is the majority vote combiner (Chapter 4). Other combiners need additional training, for example, the weighted average combiner (Chapter 5). A third class of ensembles develop the combiner during the training of the individual classifiers, for example, AdaBoost (Chapter 7). The following phrases can be found in the literature

trainable combiners – nontrainable combiners [113]
data-dependent ensembles – data-independent ensembles [114]

Data-dependent ensembles are split into implicitly dependent and explicitly dependent. The implicitly data-dependent group contains *trainable* combiners for classifier fusion where the fusion parameters do not depend on the input $\mathbf{x}$. In other words, the fusion parameters are trained before the system is used for labeling new inputs. The explicit data-dependent combiners use weights that are functions of $\mathbf{x}$. This group includes the "pure" classifier selection because the data-dependent weights can be set as a binary vector with 1 corresponding to the classifier responsible for $\mathbf{x}$ and 0 for all other classifiers.

3.3 TO TRAIN OR NOT TO TRAIN?

3.3.1 Tips for Training the Ensemble

If a large data set is available, then we have many options among which are:

- train a single (possibly complex) classifier;
- train the base classifiers on nonoverlapping training sets;

- use the pasting-small-votes idea (Chapter 7);
- evaluate the ensemble and the single classifiers very precisely (using a large testing set) so as to be able to decide what to use in practice.

While large data sets are abundant with possibilities, relatively small data sets pose a real challenge. Duin [113] points out the crucial role of the training strategy in these cases and gives the following recommendations:

1. If a single training set is used with a *nontrainable combiner*, then make sure that the base classifiers are not overtrained. In case probabilistic outputs are needed, these have to be reliably calibrated too, with as little overtraining as possible. In a way, the accuracy of the base classifiers is regarded as less important for the success of the ensemble than the reliability of the decisions and probability estimates.
2. If a single training set is used with a *trainable combiner*, then leave the base classifiers undertrained and subsequently complete the training of the combiner on the training set. Here it is assumed that the training set has a certain amount of training potential. In order to be able to be train the combiner reasonably, the base classifiers should not use up all the potential.
3. Use separate training sets for the base classifiers and for the combiners. Then the base classifiers can be overtrained on their training set. The bias will be corrected by training the combiner on the separate training set.

Dietrich et al. [115] suggest that the second training set, on which the ensemble should be trained, may be partly overlapping with the first training set used for the individual classifiers. Let R be the first training set, V be the second training set, and T be the testing set. All three sets are obtained from the available labeled set $\mathbf{Z}$ so $R \cup V \cup T = \mathbf{Z}$. If $\mathbf{Z}$ is small, the three sets might become inadequately small thereby leading to badly trained classifiers and ensemble, and unreliable estimates of their accuracies. To remedy this, the two training sets are allowed to have an overlap controlled by a parameter ρ

$$\rho = \frac{|R \cap V|}{|R|} \tag{3.1}$$

where $|\cdot|$ denotes the cardinality. For $\rho = 0$, R and V are disjoined and for $\rho = 1$, the classifiers and the ensemble are trained on a single set $R = V$. The authors found that best results were obtained for $\rho = 0.5$ compared to the two extreme values. This suggests that a compromise should be sought when the initial data set $\mathbf{Z}$ is relatively small. Another possible way of expanding the training set is the so-called *out of bag estimate* [116]. This estimate is appropriate for bagging where the training sets for the individual classifiers are sampled with replacement from the initial training set. The out-of-bag estimates are explained in Chapter 7.

Training	A	B	C	D
	B	C	D	A
	C	D	A	B
Testing	D	A	B	C

Fig. 3.5 *Standard four-fold cross-validation set-up.*

3.3.2 Idea of Stacked Generalization

Stacked generalization has been defined as a generic methodology for improving generalization in pattern classification [117]. We will regard it here as a philosophy for combining classifiers with a special emphasis on the training protocol.

Let $\mathbf{Z}$ be our data set with N points $\mathbf{z}_j \in \Re^n$ labeled in $\Omega = \{\omega_1, \ldots, \omega_c\}$. Let us partition $\mathbf{Z}$ into four disjoint subsets of roughly equal size, A, B, C, and D. Suppose that we have three classifier models D_1, D_2, D_3, and have trained each classifier according to the standard four-fold cross-validation process depicted in Figure 3.5. At the end of this training there will be four versions of each of our classifiers trained on (ABC), (BCD), (ACD), or (ABD), respectively.

The combiner is trained on a data set of size N obtained in the following way. For any data point $\mathbf{z}_j$ in subset A, we take the outputs for that point from the versions of D_1, D_2, and D_3 built on (BCD). In this way subset A has not been seen during the training of the individual classifiers. The three outputs together with the label of $\mathbf{z}_j$ form a data point in the training set for the combiner. All the points from subset B are processed by the versions of the three classifiers built on (ACD) and the outputs added to the training set for the combiner, and so on. After the combiner has been trained, the four subsets are pooled again into $\mathbf{Z}$ and D_1, D_2, and D_3 are retrained, this time on the whole of $\mathbf{Z}$. The new classifiers and the combiner are then ready for operation.

3.4 REMARKS

We might pride ourselves for working in a modern area of pattern recognition and machine learning that started about a decade ago but, in fact, combining classifiers is much older. Take for example the idea of viewing the classifier output as a new feature vector. This could be traced back to Sebestyen [9] in his book *Decision-Making Processes in Pattern Recognition*, published in 1962. Sebestyen proposes cascade machines where the output of a classifier is fed as the input of the next classifier in the sequence, and so on. In 1975 Dasarathy and Sheila [118] propose a compound classifier where the decision is switched between two different classifier models depending on where the input is located. The book by Rastrigin and Erenstein [119], published in 1981, contains what is now known as dynamic classifier selection [120]. Unfortunately, Rastrigin and Erenstein's book only reached the

Russian-speaking reader, and so did the book by Barabash, published in 1983 [121], containing interesting theoretical results about the majority vote for classifier combination.

How far have we gone? It is curious that the experts in the field hold diametrically opposite views about our current level of understanding of combining classifiers. In his invited lecture at the 3rd International Workshop on Multiple Classifier Systems, 2002, Ghosh proposes that [122]

> . . . our current understanding of ensemble-type multiclassifier systems is now quite mature . . .

And yet, in an invited book chapter, the same year, Ho states that [105]

> . . . Many of the above questions are there because we do not yet have a scientific understanding of the classifier combination mechanisms.

Ho proceeds to nominate the stochastic discrimination theory by Kleinberg [123,124] as the only consistent and theoretically sound explanation of the success of classifier ensembles, criticizing other theories as being incomplete and assumption-bound.

However, as the usual practice invariably shows, ingenious heuristic developments are the heart, the soul, and the engine of many branches of science and research. A snapshot of the opinions of the participants in the last edition of the International Workshop on Multiple Classifier Systems, MCS 2003, in Guildford, UK, suggests that the area of combining classifiers is very dynamic and active at present, and is likely to grow and expand in the nearest future.

4

Fusion of Label Outputs

4.1 TYPES OF CLASSIFIER OUTPUTS

The possible ways of combining the outputs of the L classifiers in an ensemble depend on what information we obtain from the individual members. Xu et al. [125] distinguish between three types of classifier outputs:

- *Type* 1 (*The Abstract level*). Each classifier D_i produces a class label $s_i \in \Omega$, $i = 1, \ldots, L$. Thus, for any object $\mathbf{x} \in \Re^n$ to be classified, the L classifier outputs define a vector $\mathbf{s} = [s_1, \ldots, s_L]^T \in \Omega^L$. At the abstract level, there is no information about the certainty of the guessed labels, nor are any alternative labels suggested. By definition, any classifier is capable of producing a label for $\mathbf{x}$, so the abstract level is the most universal one.
- *Type* 2 (*The Rank level*). The output of each D_i is a subset of Ω, with the alternatives ranked in order of plausibility of being the correct label [126,127]. Type 2 is especially suitable for problems with a large number of classes, for example, character/face/speaker recognition, and so on.
- *Type* 3 (*The Measurement level*). Each classifier D_i produces a c-dimensional vector $[d_{i,1}, \ldots, d_{i,c}]^T$. The value $d_{i,j}$ represents the support for the hypothesis that vector $\mathbf{x}$ submitted for classification comes from class ω_j.[14] Without loss of

[14] The outputs $d_{i,j}$ are functions of the input $\mathbf{x}$. To simplify the notation we will use just $d_{i,j}$ instead of $d_{i,j}(\mathbf{x})$.

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

generality, we can assume that the outputs contain values between 0 and 1, each classifier output spanning the space $[0,1]^c$.

We add to this list one more item:

- *Type* 0 (*The Oracle level*). The output of classifier D_i for a given $\mathbf{x}$ is only known to be either *correct* or *wrong*. We deliberately disregard the information as to which class label has been assigned. The oracle output is artificial because we can only apply it to a labeled data set. For a given data set $\mathbf{Z}$, classifier D_i produces an output vector $\mathbf{y}_i$ such that

$$y_{ij} = \begin{cases} 1, & \text{if } D_i \text{ classifies object } \mathbf{z}_j \text{ correctly,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

In this chapter we consider methods for combining *label outputs*.

4.2 MAJORITY VOTE

4.2.1 Democracy in Classifier Combination

Dictatorship and majority vote are perhaps the two oldest strategies for decision making. Day [128] reviews the evolution of the concept of consensus in a context of electoral theory. Its roots are traced back to the era of ancient Greek city states and the Roman Senate. The majority criterion became established in 1356 for the election of German kings, by 1450 was adopted for elections to the British House of Commons, and by 1500 as a rule to be followed in the House itself.

Three consensus patterns, unanimity, simple majority, and plurality, are illustrated in Figure 4.1. If we assume that black, gray, and white correspond to class labels, and the decision makers are the individual classifiers in the ensemble, the final label will be "black" for all three patterns.

Assume that the *label* outputs of the classifiers are given as c-dimensional binary vectors $[d_{i,1},\ldots,d_{i,c}]^T \in \{0,1\}^c$, $i = 1,\ldots,L$, where $d_{i,j} = 1$ if D_i labels $\mathbf{x}$ in ω_j, and 0 otherwise. The *plurality vote* will result in an ensemble decision for class ω_k if

$$\sum_{i=1}^{L} d_{i,k} = \max_{j=1}^{c} \sum_{i=1}^{L} d_{i,j} \tag{4.2}$$

Ties are resolved arbitrarily. This rule is often called in the literature *the majority vote*. It will indeed coincide with the simple majority (50 percent of the votes $+1$) in the case of two classes ($c = 2$). Xu et al. [125] suggest a thresholded plurality vote. They augment the set of class labels Ω with one more class, ω_{c+1}, for all objects for which the ensemble either fails to determine a class label with a sufficient

Fig. 4.1 *Consensus patterns in a group of 10 decision makers: unanimity, simple majority, and plurality. In all three cases the final decision of the group is "black."*

confidence or produces a tie. Thus the decision is

$$\begin{cases} \omega_k, & \text{if } \sum_{i=1}^{L} d_{i,k} \geq \alpha \cdot L, \\ \omega_{c+1}, & \text{otherwise,} \end{cases} \tag{4.3}$$

where $0 < \alpha \leq 1$. For the simple majority, we can pick α to be $\frac{1}{2} + \varepsilon$, where $0 < \varepsilon < 1/L$. When $\alpha = 1$, Eq. (4.3) becomes the *unanimity vote* rule: a decision is made for some class label if all decision makers agree on that label; otherwise the ensemble refuses to decide and assigns label ω_{c+1} to $\mathbf{x}$.

The plurality vote of Eq. (4.2), called in a wide sense "the majority vote," is the most often used rule from the majority vote group. Various studies are devoted to the majority vote for classifier combination [121,129–133].

To find out why the majority vote is one of the most popular combination schemes, we will examine its properties. Assume that:

- The number of classifiers, L, is odd.
- The probability for each classifier to give the correct class label is p for any $\mathbf{x} \in \Re^n$.
- The classifier outputs are independent; that is, for any subset of classifiers $A \subseteq \mathcal{D}, A = \{D_{i_1}, \ldots, D_{i_K}\}$, the joint probability can be decomposed as

$$P(D_{i_1} = s_{i_1}, \ldots, D_{i_K} = s_{i_K}) \;=\; P(D_{i_1} = s_{i_1}) \times \cdots \times P(D_{i_K} = s_{i_K}) \tag{4.4}$$

where s_{i_j} is the label output of classifier $D_{i,j}$.

According to Eq. (4.2), the majority vote will give an accurate class label if at least $\lfloor L/2 \rfloor + 1$ classifiers give correct answers ($\lfloor a \rfloor$ denotes the "floor," that is,

TABLE 4.1 Tabulated Values of the Majority Vote Accuracy of L Independent Classifiers with Individual Accuracy p.

	$L = 3$	$L = 5$	$L = 7$	$L = 9$
$p = 0.6$	0.6480	0.6826	0.7102	0.7334
$p = 0.7$	0.7840	0.8369	0.8740	0.9012
$p = 0.8$	0.8960	0.9421	0.9667	0.9804
$p = 0.9$	0.9720	0.9914	0.9973	0.9991

the nearest integer smaller than a).[15] Then the accuracy of the ensemble is

$$P_{\text{maj}} = \sum_{m=\lfloor L/2 \rfloor + 1}^{L} \binom{L}{m} p^m (1-p)^{L-m} \tag{4.5}$$

The probabilities of correct classification of the ensemble for $p = 0.6$, 0.7, 0.8, and 0.9, and $L = 3$, 5, 7, and 9, are displayed in Table 4.1.

The following result is also known as the Condorcet Jury Theorem (1785) [134]:

1. If $p < 0.5$, then P_{maj} in Eq. (4.5) is monotonically increasing and

$$P_{\text{maj}} \longrightarrow 1 \quad \text{as} \quad L \longrightarrow \infty \tag{4.6}$$

2. If $p < 0.5$, then P_{maj} at Eq. (4.5) is monotonically decreasing and

$$P_{\text{maj}} \longrightarrow 0 \quad \text{as} \quad L \longrightarrow \infty \tag{4.7}$$

3. If $p = 0.5$, then $P_{\text{maj}} = 0.5$ for any L.

This result supports the intuition that we can expect improvement over the individual accuracy p only when p is higher than 0.5. Lam and Suen [131] proceed to analyze the case of even L and the effect on the ensemble accuracy of adding or removing classifiers.

Shapley and Grofman [134] note that the result is valid even for unequal p, provided the distribution of the individual accuracies p_i is symmetrical about the mean.

Example: Majority and Unanimity in Medical Diagnostics. An accepted practice in medicine is to confirm the diagnosis by several (supposedly independent) tests. Lachenbruch [135] studies the unanimity and majority rules on a sequence of three tests for HIV diagnosis.

[15] Notice that the majority (50 percent + 1) is necessary and sufficient for a correct decision in the case of two classes, and is sufficient but not necessary for $c > 2$. Thus the "true" accuracy of an ensemble using plurality when $c > 2$ could be greater than the majority vote accuracy.

TABLE 4.2 Unanimity and Majority Schemes for Three Independent Consecutive Tests.

Method	Unanimity		Majority	
Decisions	(+)	(−)	(+)	(−)
The consecutive readings	+++	−	++	−−
		+ −	+−+	+−−
		++ −	−++	−+−

Sensitivity and Specificity. These are the two most important characteristics of a medical test. Sensitivity (denoted by U) is the probability that the test procedure declares an affected individual affected (probability of a *true positive*). Specificity (denoted by V) is the probability that the test procedure declares an unaffected individual unaffected (probability of a *true negative*).[16]

Let T denote the positive test result, and A denote "affected." Then $U = P(T|A)$ and $V = P(\bar{T}|\bar{A})$. We regard the test as an individual classifier with accuracy $p = U \cdot P(A) + V \cdot [1 - P(A)]$, where $P(A)$ is the probability for the occurrence of the disease among the examined individuals, or the *prevalence* of the disease. In testing for HIV, a unanimous positive result from three tests is required to declare the individual affected [135]. Since the tests are applied one at a time, encountering the first negative result will cease the procedure. Another possible combination is the majority vote, which will stop if the first two readings agree or otherwise take the third reading to resolve the tie. Table 4.2 shows the outcomes of the tests and the overall decision for the unanimity and majority rules.

Assume that the three tests are applied independently and all have the same sensitivity u and specificity v. Then the sensitivity and the specificity of the procedure with the unanimity vote become

$$\begin{aligned} U_{\text{una}} &= u^3 \\ V_{\text{una}} &= 1 - (1 - v)^3 \end{aligned} \tag{4.8}$$

For the majority vote,

$$\begin{aligned} U_{\text{maj}} &= u^2 + 2u^2(1 - u) = u^2(3 - 2u) \\ V_{\text{maj}} &= v^2(3 - 2v) \end{aligned} \tag{4.9}$$

For $0 < u < 1$ and $0 < v < 1$, by simple algebra we obtain

$$U_{\text{una}} < u \quad \text{and} \quad V_{\text{una}} > v \tag{4.10}$$

[16] In social sciences, for example, sensitivity translates to "convicting the guilty" and specificity to "freeing the innocent" [134].

and

$$U_{\text{maj}} > u \quad \text{and} \quad V_{\text{maj}} > v \tag{4.11}$$

Thus, there is a certain gain on both sensitivity and specificity if majority vote is applied. Therefore the combined accuracy $P_{\text{maj}} = U \cdot P(A) + V \cdot [1 - P(A)]$ is also higher than the accuracy of a single test $p = u \cdot P(A) + v \cdot [1 - P(A)]$. For the unanimity rule, there is a substantial increase of specificity at the expense of decreased sensitivity. To illustrate this point, consider the ELISA test used for diagnosing HIV. According to Ref. [135], this test has been reported to have sensitivity $u = 0.95$ and specificity $v = 0.99$. Then

$$U_{\text{una}} \approx 0.8574 \qquad U_{\text{maj}} \approx 0.9928$$
$$V_{\text{una}} \approx 1.0000 \qquad V_{\text{maj}} \approx 0.9997$$

The sensitivity of the unanimity scheme is dangerously low. This means that the chance of an affected individual being misdiagnosed as unaffected is above 14 percent. There are different ways to remedy this. One possibility is to apply a more expensive and more accurate second test in case ELISA gave a positive result, for example, the Western blot test, for which $u = v = 0.99$ [135].

4.2.2 Limits on the Majority Vote Accuracy: An Example

Let $\mathcal{D} = \{D_1, D_2, D_3\}$ be an ensemble of three classifiers with the same individual probability of correct classification $p = 0.6$. Suppose that there are 10 objects in a hypothetical data set, and that each classifier labels correctly exactly six of them. Each classifier output is recorded as correct (1) or wrong (0). Given these requirements, *all* possible combinations of distributing 10 elements into the eight combinations of outputs of the three classifiers are shown in Table 4.3. The penultimate column shows the majority vote accuracy of each of the 28 possible combinations. It is obtained as the proportion (out of 10 elements) of the sum of the entries in columns 111, 101, 011, and 110 (two or more correct votes). The rows of the table are ordered by the majority vote accuracy. To clarify the entries in Table 4.3, consider as an example the first row. The number 3 in the column under the heading 101, means that exactly three objects are correctly recognized by D_1 and D_3 (the first and the third 1s of the heading) and misclassified by D_2 (the zero in the middle).

The table offers a few interesting facts:

- There is a case where the majority vote produces 90 percent correct classification. Although purely hypothetical, this vote distribution is *possible* and offers a dramatic increase over the individual rate $p = 0.6$.

TABLE 4.3 All Possible Combinations of Correct/Incorrect Classification of 10 Objects by Three Classifiers so that Each Classifier Recognizes Exactly Six Objects.

No.	111 *a*	101 *b*	011 *c*	001 *d*	110 *e*	100 *f*	010 *g*	000 *h*	P_{maj}	$P_{\text{maj}} - p$
Pattern of success										
1	**0**	**3**	**3**	**0**	**3**	**0**	**0**	**1**	**0.9**	**0.3**
2	2	2	2	0	2	0	0	2	0.8	0.2
3	1	2	2	1	3	0	0	1	0.8	0.2
4	0	2	3	1	3	1	0	0	0.8	0.2
5	0	2	2	2	4	0	0	0	0.8	0.2
6	4	1	1	0	1	0	0	3	0.7	0.1
7	3	1	1	1	2	0	0	2	0.7	0.1
8	2	1	2	1	2	1	0	1	0.7	0.1
9	2	1	1	2	3	0	0	1	0.7	0.1
10	1	2	2	1	2	1	1	0	0.7	0.1
11	1	1	2	2	3	1	0	0	0.7	0.1
12	1	1	1	3	4	0	0	0	0.7	0.1
Identical classifiers										
13	6	0	0	0	0	0	0	4	0.6	0.0
14	5	0	0	1	1	0	0	3	0.6	0.0
15	4	0	1	1	1	1	0	2	0.6	0.0
16	4	0	0	2	2	0	0	2	0.6	0.0
17	3	1	1	1	1	1	1	1	0.6	0.0
18	3	0	1	2	2	1	0	1	0.6	0.0
19	3	0	0	3	3	0	0	1	0.6	0.0
20	2	1	1	2	2	1	1	0	0.6	0.0
21	2	0	2	2	2	2	0	0	0.6	0.0
22	2	0	1	3	3	1	0	0	0.6	0.0
23	2	0	0	4	4	0	0	0	0.6	0.0
24	5	0	0	1	0	1	1	2	0.5	−0.1
25	4	0	0	2	1	1	1	1	0.5	−0.1
26	3	0	1	2	1	2	1	0	0.5	−0.1
27	3	0	0	3	2	1	1	0	0.5	−0.1
Pattern of failure										
28	**4**	**0**	**0**	**2**	**0**	**2**	**2**	**0**	**0.4**	**−0.2**

- On the other hand, the majority vote is not guaranteed to do better than a single member of the team. The combination in the bottom row has a majority vote accuracy of 0.4.

The best and the worst possible cases illustrated above are named "the pattern of success" and the "pattern of failure" [136] and are detailed next.

4.2.3 Patterns of Success and Failure

Consider two classifiers D_i and D_k, and a 2×2 table of probabilities that summarizes their combined outputs as in Table 4.4.

TABLE 4.4 The 2 × 2 Relationship Table with Probabilities.

	D_k correct (1)	D_k wrong (0)
D_i correct (1)	a	b
D_i wrong (0)	c	d

Total, $a + b + c + d = 1$.

TABLE 4.5 The Probabilities in Two 2-Way Tables Illustrating a Three-Classifier Voting Team.

D_3 correct (1)			D_3 wrong (0)		
	$D_2 \rightarrow$			$D_2 \rightarrow$	
$D_1 \downarrow$	1	0	$D_1 \downarrow$	1	0
1	a	b	1	e	f
0	c	d	0	g	h

The three-classifier problem from the previous section can be visualized using two pairwise tables as in Table 4.5. For this case,

$$a + b + c + d + e + f + g + h = 1 \tag{4.12}$$

The probability of correct classification of the majority vote of the three classifiers is (two or more correct)

$$P_{\text{maj}} = a + b + c + e \tag{4.13}$$

All three classifiers have the same individual accuracy p, which brings in the following three equations:

$$\begin{aligned} a + b + e + f &= p, && D_1 \text{ correct} \\ a + c + e + g &= p, && D_2 \text{ correct} \\ a + b + c + d &= p, && D_3 \text{ correct} \end{aligned} \tag{4.14}$$

Maximizing P_{maj} in Eq. (4.13) subject to conditions (4.12), (4.14), and $a, b, c, d, e, f, g, h \geq 0$, for $p = 0.6$, we obtain $P_{\text{maj}} = 0.9$ with the pattern highlighted in Table 4.3: $a = d = f = g = 0$, $b = c = e = 0.3$, $h = 0.1$. This example, optimal for three classifiers, indicates the possible characteristics of the best combination of L classifiers. The "pattern of success" and "pattern of failure" defined later follow the same intuition although we do not include a formal proof for their optimality.

Consider the ensemble $\mathcal{D}$ of L classifiers, each with accuracy p (assume L is odd). For the majority vote to give a correct answer we need $\lfloor L/2 \rfloor + 1$ or more of the classifiers to be correct. Intuitively, the best improvement over the individual accuracy will be achieved when exactly $\lfloor L/2 \rfloor + 1$ votes are correct. Any extra correct vote for the same $\mathbf{x}$ will be "wasted" because it is not needed to give the correct class label. Correct votes that participate in combinations not leading to a correct overall vote are also "wasted". To use the above idea we make the following definition.

The "*pattern of success*" is a distribution of the L classifier outputs for the ensemble $\mathcal{D}$ such that:

1. The probability of any combination of $\lfloor L/2 \rfloor + 1$ correct and $\lfloor L/2 \rfloor$ incorrect votes is α.
2. The probability of all L votes being incorrect is γ.
3. The probability of any other combination is zero.

For $L = 3$, the two-table expression of the pattern of success is shown in Table 4.6.

Here no votes are wasted; the only combinations that occur are where all classifiers are incorrect or exactly $\lfloor L/2 \rfloor + 1$ are correct. To simplify notation, let $l = \lfloor L/2 \rfloor$. The probability of a correct majority vote (P_{maj}) for the pattern of success is the sum of the probabilities of each correct majority vote combination. Each such combination has probability α. There are $\binom{L}{l+1}$ ways of having $l + 1$ correct out of L classifiers. Therefore

$$P_{\text{maj}} = \binom{L}{l+1}\alpha \tag{4.15}$$

The pattern of success is only possible when $P_{\text{maj}} \leq 1$; that is, when

$$\alpha \leq \frac{1}{\binom{L}{l+1}} \tag{4.16}$$

TABLE 4.6 The Pattern of Success.

D_3 correct (1)			D_3 wrong (0)		
	$D_2 \rightarrow$			$D_2 \rightarrow$	
$D_1 \downarrow$	1	0	$D_1 \downarrow$	1	0
1	0	α	1	α	0
0	α	0	0	0	$\gamma = 1 - 3\alpha$

To relate the individual accuracies p to α and P_{maj}, consider the following argument. In the pattern of success, if D_i gives a correct vote, then the remaining $L-1$ classifiers must give l correct votes. There are $\binom{L-1}{l}$ ways in which the remaining $L-1$ classifiers can give l correct votes, each with probability α. So the overall accuracy p of D_i is

$$p = \binom{L-1}{l}\alpha \tag{4.17}$$

Expressing α from Eq. (4.17) and substituting in Eq. (4.15) gives

$$P_{\text{maj}} = \frac{pL}{l+1} = \frac{2pL}{L+1} \tag{4.18}$$

Feasible patterns of success have $P_{\text{maj}} \leq 1$, so Eq. (4.18) requires

$$p \leq \frac{L+1}{2L} \tag{4.19}$$

If $p > L + 1/2L$ then $P_{\text{maj}} = 1$ can be achieved, but there is an excess of correct votes. The improvement over the individual p will not be as large as for the pattern of success but the majority vote accuracy will be 1 anyway. The final formula for P_{maj} is

$$P_{\text{maj}} = \min\left\{1, \frac{2pL}{L+1}\right\} \tag{4.20}$$

The worst possible behavior of an ensemble of L classifiers each with accuracy p is described by the pattern of failure.

The "*pattern of failure*" is a distribution of the L classifier outputs for the ensemble $\mathcal{D}$ such that:

1. The probability of any combination of $\lfloor L/2 \rfloor$ correct and $\lfloor L/2 \rfloor + 1$ incorrect votes is β.
2. The probability of all L votes being correct is δ.
3. The probability of any other combination is zero.

For $L = 3$, the two-table expression of the pattern of failure is shown in Table 4.7.

The worst scenario is when the correct votes are wasted, that is, grouped in combinations of exactly l out of L correct (one short for the majority to be correct). The excess of correct votes needed to make up the individual p are also wasted by all the votes being correct together, while half of them plus one will suffice.

TABLE 4.7 The Pattern of Failure.

D_3 correct (1)			D_3 wrong (0)		
	$D_2 \rightarrow$			$D_2 \rightarrow$	
$D_1 \downarrow$	1	0	$D_1 \downarrow$	1	0
1	$\delta = 1 - 3\beta$	0	1	0	β
0	0	β	0	β	0

The probability of a correct majority vote (P_{maj}) is δ. As there are $\binom{L}{l}$ ways of having l correct out of L classifiers, each with probability β, then

$$P_{\text{maj}} = \delta = 1 - \binom{L}{l}\beta \tag{4.21}$$

If D_i gives a correct vote then either all the remaining classifiers are correct (probability δ) or exactly $l-1$ are correct out of the $L-1$ remaining classifiers. For the second case there are $\binom{L-1}{l-1}$ ways of getting this, each with probability β. To get the overall accuracy p for classifier D_i we sum the probabilities of the two cases

$$p = \delta + \binom{L-1}{l-1}\beta \tag{4.22}$$

Combining Eqs. (4.21) and (4.22) gives

$$P_{\text{maj}} = \frac{pL - l}{l+1} = \frac{(2p-1)L + 1}{L+1} \tag{4.23}$$

For values of individual accuracy $p > 0.5$, the pattern of failure is always possible.

Matan [137] gives tight upper and lower bounds of the majority vote accuracy in the case of unequal individual accuracies. Suppose that classifier D_i has accuracy p_i, and $\{D_1, \ldots, D_L\}$ are arranged so that $p_1 \leq p_2 \cdots \leq p_L$. Let $k = l + 1 = (L+1)/2$. Matan proves that

1. The upper bound of the majority vote accuracy of the ensemble is

$$\max P_{\text{maj}} = \min\{1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(1)\} \tag{4.24}$$

where

$$\Sigma(m) = \frac{1}{m}\sum_{i=1}^{L-k+m} p_i, \qquad m = 1, \ldots, k \tag{4.25}$$

2. The lower bound of the majority vote accuracy of the ensemble is

$$\min P_{\text{maj}} = \max\{0, \xi(k), \xi(k-1), \ldots, \xi(1)\} \tag{4.26}$$

where

$$\xi(m) = \frac{1}{m} \sum_{i=k-m+1}^{L} p_i - \frac{L-k}{m}, \qquad m = 1, \ldots, k \tag{4.27}$$

Example: Matan's Limits on the Majority Vote Accuracy. Let $\mathcal{D} = \{D_1, \ldots, D_5\}$ with accuracies (0.56, 0.58, 0.60, 0.60, 0.62), respectively. For this ensemble $k = (5+1)/2 = 3$. To find the upper bound of the majority vote accuracy of this team, form the sums $\Sigma(m)$ for $m = 1, 2, 3$

$$\begin{aligned}
\Sigma(1) &= 0.56 + 0.58 + 0.60 = 1.74 \\
\Sigma(2) &= \frac{1}{2}(0.56 + 0.58 + 0.60 + 0.60) = 1.17 \\
\Sigma(3) &= \frac{1}{3}(0.56 + 0.58 + 0.60 + 0.60 + 0.62) = 0.99
\end{aligned} \tag{4.28}$$

Then

$$\max P_{\text{maj}} = \min\{1, 1.74, 1.17, 0.99\} = 0.99 \tag{4.29}$$

For the lower bound,

$$\begin{aligned}
\xi(1) &= 0.60 + 0.60 + 0.62 - (5-3) = -0.18 \\
\xi(2) &= \frac{1}{2}(0.58 + 0.60 + 0.60 + 0.62) - \frac{5-3}{2} = 0.20 \\
\xi(3) &= \frac{1}{3}(0.56 + 0.58 + 0.60 + 0.60 + 0.62) - \frac{5-3}{3} = 0.32
\end{aligned} \tag{4.30}$$

Then

$$\min P_{\text{maj}} = \max\{0, -0.18, 0.20, 0.32\} = 0.32 \tag{4.31}$$

The range of possible results from the majority vote across $\mathcal{D}$ is wide, so without more knowledge about how the classifiers are related to each other we can only guess within this range. If we assume that the classifier outputs are independent, then $P_{\text{maj}} = 0.67$, which indicates that there is much more to be achieved from the majority vote than what independent outputs can offer.

Matan's result leads to the pattern of success and the pattern of failure as the upper and the lower bounds respectively, for $p_1 = \cdots = p_L = p$. Demirekler and Altincay [138] and Ruta and Gabrys [133] give further insights into the behavior of the two limit patterns.

Hierarchical majority voting ensembles have been found very promising [133,134,137]. There is a potential gain in accuracy, but this has only been shown by construction examples.

4.3 WEIGHTED MAJORITY VOTE

If the classifiers in the ensemble are not of identical accuracy, then it is reasonable to attempt to give the more competent classifiers more power in making the final decision. The label outputs can be represented as degrees of support for the classes in the following way

$$d_{i,j} = \begin{cases} 1, & \text{if } D_i \text{ labels } \mathbf{x} \text{ in } \omega_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.32}$$

The discriminant function for class ω_j obtained through weighted voting is

$$g_j(\mathbf{x}) = \sum_{i=1}^{L} b_i d_{i,j} \tag{4.33}$$

where b_i is a coefficient for classifier D_i. Thus the value of the discriminant function (4.33) will be the sum of the coefficients for these members of the ensemble whose output for $\mathbf{x}$ is ω_j.

Example: Assigning Weights to the Classifiers. Consider a team of three classifiers D_1, D_2, and D_3 with accuracies 0.6, 0.6, and 0.7, respectively, and with independent outputs. Then

$$P_{\text{maj}} = 0.6^2 \times 0.3 + 2 \times 0.4 \times 0.6 \times 0.7 + 0.6^2 \times 0.7 = 0.6960 \tag{4.34}$$

Clearly, it will be better if we remove D_1 and D_2 and reduce the ensemble to the single and more accurate classifier D_3. We introduce weights or coefficients of importance b_i, $i = 1, 2, 3$, and rewrite Eq. (4.2) as: choose class label ω_k if

$$\sum_{i=1}^{L} b_i d_{i,k} = \max_{j=1}^{c} \sum_{i=1}^{L} b_i d_{i,j} \tag{4.35}$$

For convenience we normalize the coefficients so that

$$\sum_{i=1}^{c} b_j = 1 \tag{4.36}$$

Assigning $b_1 = b_2 = 0$ and $b_3 = 1$, we get rid of D_1 and D_2, leading to $P_{\text{maj}} = p_3 = 0.7$. In fact, any set of weights that makes D_3 the dominant classifier will yield the same P_{maj}, for example, $b_3 > 0.5$ and any b_1 and b_2 satisfying Eq. (4.36).

In the above example the weighted voting did not improve on the single best classifier in the team even for independent classifiers. The following example shows that, in theory, the weighting might lead to a result better than both the single best member of the team and the simple majority.

Example: Improving the Accuracy by Weighting. Consider an ensemble of five classifiers $D_1, \ldots, D_5$ with accuracies (0.9, 0.9, 0.6, 0.6, 0.6).[17] If the classifiers are independent, the majority vote accuracy (at least three correct out of five votes) is

$$\begin{aligned} P_{\text{maj}} &= 3 \times 0.9^2 \times 0.4 \times 0.6 + 0.6^3 + 6 \times 0.9 \times 0.1 \times 0.6^2 \times 0.4 \\ &\approx 0.877 \end{aligned} \tag{4.37}$$

Assume now that the weights given to the voters are $(1/3, 1/3, 1/9, 1/9, 1/9)$. Then the two more competent classifiers agreeing will be enough to make the decision because the score for the class label they agree upon will become $2/3$. If they disagree, that is, one is correct and one is wrong, the vote of the team will be decided by the majority of the remaining three classifiers. Then the accuracy for the weighted voting will be

$$\begin{aligned} P^w_{\text{maj}} &= 0.9^2 + 2 \times 3 \times 0.9 \times 0.1 \times 0.6^2 \times 0.4 + 2 \times 0.9 \times 0.1 \times 0.6^3 \\ &\approx 0.927 \end{aligned} \tag{4.38}$$

Again, any set of weights that satisfy Eq. (4.36) and make the first two classifiers prevail when they agree, will lead to the same outcome.

One way to select the weights for the classifiers is formalized through the following theorem.

Theorem 4.1. Consider an ensemble of L *independent*[18] classifiers $D_1, \ldots, D_L$, with individual accuracies $p_1, \ldots, p_L$. The outputs are combined by the weighted majority vote (4.35). Then the accuracy of the ensemble (P^w_{maj}) is maximized by assigning weights

$$b_i \propto \log \frac{p_i}{1 - p_i} \tag{4.39}$$

[17] After Ref. [134].

[18] The conditional independence is considered, that is, $P(\mathbf{s}|\omega_j) = \prod_{i=1}^{L} P(s_i|\omega_j)$.

Proof. Denote by $s = [s_1, \ldots, s_L]^T$ the vector with the label output of the ensemble, where $s_i \in \Omega$ is the label suggested for $\mathbf{x}$ by classifier D_i. A Bayes-optimal set of discriminant functions based on the outputs of the L classifiers is

$$g_j(\mathbf{x}) = \log P(\omega_j)P(\mathbf{s}|\omega_j), \qquad j = 1, \ldots, c \tag{4.40}$$

From the conditional independence

$$g_j(\mathbf{x}) = \log\left[P(\omega_j)\prod_{i=1}^{L} P(s_i|\omega_j)\right] \tag{4.41}$$

$$= \log P(\omega_j) + \log \prod_{i,s_i=\omega_j} P(s_i|\omega_j) \prod_{i,s_i\neq\omega_j} P(s_i|\omega_j) \tag{4.42}$$

$$= \log P(\omega_j) + \log \prod_{i,s_i=\omega_j} p_i \prod_{i,s_i\neq\omega_j} (1-p_i) \tag{4.43}$$

$$= \log P(\omega_j) + \log \prod_{i,s_i=\omega_j} \frac{p_i(1-p_i)}{1-p_i} \prod_{i,s_i\neq\omega_j} (1-p_i) \tag{4.44}$$

$$= \log P(\omega_j) + \log \prod_{i,s_i=\omega_j} \frac{p_i}{1-p_i} \prod_{i=1}^{L} (1-p_i) \tag{4.45}$$

$$= \log P(\omega_j) + \sum_{i,s_i=\omega_j} \log \frac{p_i}{1-p_i} + \sum_{i=1}^{L} \log(1-p_i) \tag{4.46}$$

The last term in this summation does not depend on the class label j; therefore we can reduce the discriminant function to

$$g_j(\mathbf{x}) = \log P(\omega_j) + \sum_{i=1}^{L} d_{i,j} \log \frac{p_i}{1-p_i} \tag{4.47}$$

■

Note that assigning the weights to the classifiers is not sufficient for guaranteeing the minimum classification errors. The prior probabilities for the classes have to be taken into account too.

Results similar to that of the above theorem have been derived independently by several researchers in different fields of science such as democracy studies, pattern recognition and automata theory, leading to the earliest reference [139] according to Refs. [121,134]. Curiously, the optimal weights do not take into account the performance of other members of the team but only magnify the relevance of the individual classifier based on its accuracy.

4.4 NAIVE BAYES COMBINATION

This scheme assumes that the classifiers are mutually independent given a class label (conditional independence). This is the reason why it is called "independence model" [140], "naive Bayes," "simple Bayes" [141] and even "idiot's Bayes" [15,29]. Sometimes the first adjective is skipped and the combination method is named just "*Bayes* combination."

Denote by $P(s_j)$ the probability that classifier D_j labels $\mathbf{x}$ in class $s_j \in \Omega$. The conditional independence allows for the following representation

$$P(\mathbf{s}|\omega_k) = P(s_1, s_2, \ldots, s_L|\omega_k) = \prod_{i=1}^{L} P(s_i|\omega_k) \tag{4.48}$$

Then the posterior probability needed to label $\mathbf{x}$ is

$$\begin{aligned} P(\omega_k|\mathbf{s}) &= \frac{P(\omega_k)P(\mathbf{s}|\omega_k)}{P(\mathbf{s})} \\ &= \frac{P(\omega_k)\prod_{i=1}^{L} P(s_i|\omega_k)}{P(\mathbf{s})}, \qquad k = 1, \ldots, c \end{aligned} \tag{4.49}$$

The denominator does not depend on ω_k and can be ignored, so the support for class ω_k can be calculated as

$$\mu_k(\mathbf{x}) \propto P(\omega_k) \prod_{i=1}^{L} P(s_i|\omega_k) \tag{4.50}$$

The practical implementation of the naive Bayes (NB) method on a data set $\mathbf{Z}$ with cardinality N is explained below. For each classifier D_i, a $c \times c$ confusion matrix CM^i is calculated by applying D_i to the training data set. The (k, s)th entry of this matrix, $cm^i_{k,s}$ is the number of elements of the data set whose true class label was ω_k, and were assigned by D_i to class ω_s. By N_s we denote the total number of elements of $\mathbf{Z}$ from class ω_s. Taking $cm^i_{k,s_i}/N_k$ as an estimate of the probability $P(s_i|\omega_k)$, and N_k/N as an estimate of the prior probability for class ω_s, Eq. (4.50) is equivalent to

$$\mu_k(\mathbf{x}) \propto \frac{1}{N_k^{L-1}} \prod_{i=1}^{L} cm^i_{k,s_i} \tag{4.51}$$

Example: Naive Bayes Combination. Consider a problem with $L = 2$ classifiers, D_1 and D_2, and $c = 3$ classes. Let the number of training data points be $N = 20$.

From these, let eight be from ω_1, nine from ω_2, and three from ω_3. Suppose the following confusion matrices have been obtained for the two classifiers

$$CM^1 = \begin{bmatrix} 6 & 2 & 0 \\ 1 & 8 & 0 \\ 1 & 0 & 2 \end{bmatrix} \quad \text{and} \quad CM^2 = \begin{bmatrix} 4 & 3 & 1 \\ 3 & 5 & 1 \\ 0 & 0 & 3 \end{bmatrix} \tag{4.52}$$

Assume $D_1(\mathbf{x}) = s_1 = \omega_2$ and $D_2(\mathbf{x}) = s_2 = \omega_1$ for the input $\mathbf{x} \in \Re^n$. Using Eq. (4.51)

$$\begin{aligned} \mu_1(\mathbf{x}) &\propto \frac{1}{8} \times 2 \times 4 = 1 \\ \mu_2(\mathbf{x}) &\propto \frac{1}{9} \times 8 \times 3 = \frac{8}{3} \approx 2.67 \\ \mu_3(\mathbf{x}) &\propto \frac{1}{3} \times 0 \times 0 = 0 \end{aligned} \tag{4.53}$$

As $\mu_2(\mathbf{x})$ is the highest of the three values, the maximum membership rule will label $\mathbf{x}$ in ω_2.

Notice that a zero as an estimate of $P(s_i|\omega_k)$ automatically nullifies $\mu_k(\mathbf{x})$ regardless of the rest of the estimates. Titterington et al. [140] consider the naive Bayes classifier for independent categorical features. They discuss several modifications of the estimates to account for the possible zeros. The formula they use, rewritten for the naive Bayes combination is

$$P(\mathbf{s}|\omega_k) \propto \left\{ \prod_{i=1}^{L} \frac{cm^i_{k,s_i} + 1/c}{N_k + 1} \right\}^B \tag{4.54}$$

where N_k is the number of elements in the training set $\mathbf{Z}$ from class ω_k and B is a constant.[19]

Example: Naive Bayes Combination with a Correction for Zeros. Take the 20-point data set and the confusion matrices CM^1 and CM^2 from the previous example. Using Eq. (4.54), the estimates of the class-conditional pmfs for the values $s_1 = \omega_2$,

[19] Titterington et al. [140] used $B = 1$, 0.8, and 0.5.

$s_2 = \omega_1$, and $B = 1$ are

$$\begin{aligned}
\mu_1(\mathbf{x}) &\propto \frac{N_1}{N} \times \left(\frac{cm^1_{1,2} + \frac{1}{3}}{N_1 + 1}\right)\left(\frac{cm^2_{1,1} + \frac{1}{3}}{N_1 + 1}\right) \\
&= \frac{8}{20} \times \left(\frac{2 + \frac{1}{3}}{8 + 1}\right)\left(\frac{4 + \frac{1}{3}}{8 + 1}\right) \approx 0.050 \\
\mu_2(\mathbf{x}) &\propto \frac{N_2}{N} \times \left(\frac{cm^1_{2,2} + \frac{1}{3}}{N_2 + 1}\right)\left(\frac{cm^2_{2,1} + \frac{1}{3}}{N_2 + 1}\right) \\
&= \frac{9}{20} \times \left(\frac{8 + \frac{1}{3}}{9 + 1}\right)\left(\frac{3 + \frac{1}{3}}{9 + 1}\right) \approx 0.250 \\
\mu_3(\mathbf{x}) &\propto \frac{N_3}{N} \times \left(\frac{cm^1_{3,2} + \frac{1}{3}}{N_3 + 1}\right)\left(\frac{cm^2_{3,1} + \frac{1}{3}}{N_3 + 1}\right) \\
&= \frac{3}{20} \times \left(\frac{0 + \frac{1}{3}}{3 + 1}\right)\left(\frac{0 + \frac{1}{3}}{3 + 1}\right) \approx 0.001
\end{aligned} \tag{4.55}$$

Again, label ω_2 will be assigned to $\mathbf{x}$. Notice that class ω_3 has now a small non-zero support.

The Bayes classifier has been found to be surprisingly accurate and efficient in many experimental studies. The surprise comes from the fact that the entities being combined are seldom independent. Thus the independence assumption is nearly always violated, sometimes severely. However, it turns out that the classifier performance is quite robust, even in the case of dependence. Even more, attempts to amend the naive Bayes by including estimates of some dependencies do not always pay off [141].

4.5 MULTINOMIAL METHODS

In this group of methods we estimate the posterior probabilities $P(\omega_k|\mathbf{s})$, for all $k = 1, \ldots, c$ and every combination of votes $\mathbf{s} \in \Omega^L$. The highest posterior probability determines the class label for $\mathbf{s}$. Then, given an $\mathbf{x} \in \Re^n$, first the labels $s_1, \ldots, s_L$ are assigned by the classifiers in the ensemble $\mathcal{D}$, and then the final label is retrieved for $\mathbf{s} = [s_1, \ldots, s_L]^T$.

4.5.1 Behavior Knowledge Space Method

Behavior knowledge space (BKS) is a fancy name for the multinomial combination. The label vector $\mathbf{s}$ gives an index to a cell in a look-up table (the BKS table) [142]. The table is designed using a labeled data set $\mathbf{Z}$. Each $\mathbf{z}_j \in \mathbf{Z}$ is placed in the cell

indexed by the **s** for that object. The number of elements in each cell are tallied and the most representative class label is selected for this cell. The highest score corresponds to the highest estimated posterior probability $P(\omega_k|\mathbf{s})$. Ties are resolved arbitrarily. The empty cells are labelled in some appropriate way. For example, we can choose a label at random or use the result from a majority vote between the elements of **s**.

Example: BKS Combination Method. Consider a problem with three classifiers and two classes. Assume that D_1, D_2, and D_3 produce output $(s_1, s_2, s_3) = (\omega_2, \omega_1, \omega_2)$. Suppose that there are 100 objects in $\mathbf{Z}$ for which this combination of labels occurred: 60 having label ω_1, and 40 having label ω_2. Hence the table cell indexed by $(\omega_2, \omega_1, \omega_2)$ will contain label ω_1 no matter that the majority of the classifiers suggest otherwise.

To have a reliable multinomial combiner, the data set should be large. The BKS combination method is often overtrained: it works very well on the training data but poorly on the testing data.

4.5.2 Wernecke's Method

Wernecke's combination method (WER) is similar to BKS and aims at reducing overtraining. It also uses a look-up table with labels. The difference is that in constructing the table, Wernecke [143] considers the 95 percent confidence intervals of the frequencies in each cell. If there is overlap between the intervals, the prevailing class is not considered dominating enough for labeling the cell. The "least wrong" classifier among the L members of the team is identified by calculating L estimates of the probability $P[error \text{ and } D_i(\mathbf{x}) = s_i]$. Then the classifier with the smallest probability is authorized to label the cell.

The confidence intervals are calculated according to the following assumptions and procedures. Let $k_1, \ldots, k_c$ be the number of training data points from classes $\omega_1, \ldots, \omega_c$ in the cell indexed by classifier outputs $\mathbf{s} = (s_1, \ldots, s_L)$, $s_i \in \Omega$. The BKS combiner would label the cell right away as ω_j, where $j = \arg\max_i k_i$. In case of a tie, any of the tied class labels could be picked. In WER, if there is a tie for the maximum, there is no need to calculate any confidence intervals: the least wrong classifier must be found that minimizes $P[error \text{ and } D_i(\mathbf{x}) = s_i]$. If there is a clear winner ω_j, the dominance of this class over the rest of the classes has to be established. According to Wernecke [143], we assume that k_i are binomially distributed. To calculate the 95 percent confidence intervals (CI), we use either the Normal approximation of the Binomial distribution or the Chebyshev inequality [25].

4.5.2.1 Normal Approximation of the Binomial Distribution. To go forward with this approximation and calculate the 95 percent CI for k_j [denoted

$\text{CI}(\omega_j, 95)$] the following (rule of thumb) must be true:

$$k = \sum_{i=1}^{c} k_i \geq 30$$
$$k_j \geq 5 \tag{4.56}$$
$$k - k_j \geq 5$$

Then, taking into account the continuity correction, $\text{CI}(\omega_j, 95)$ is calculated as

$$\text{CI}(\omega_j, 95) = \left[k_j - 1.96\sqrt{\frac{k_j(k-k_j)}{k}} + \frac{1}{2},\ k_j + 1.96\sqrt{\frac{k_j(k-k_j)}{k}} - \frac{1}{2} \right] \tag{4.57}$$

Example: Confidence Interval Calculation for Normal Approximation of the Binomial Distribution. Take again the outputs of D_1, D_2, and D_3 to be $\mathbf{s} = (s_1, s_2, s_3) = (\omega_2, \omega_1, \omega_2)$. Suppose there are 32 objects in the cell labeled by $\mathbf{s}$: 20 from ω_1, and 12 from ω_2. To calculate $\text{CI}(\omega_1, 95)$, first check the conditions (4.56): $32 \geq 30$; $20 \geq 5$; $32-20 \geq 5$. Then

$$\text{CI}(\omega_1, 95) = \left[20 - 1.96\sqrt{\frac{20(32-20)}{32}} + \frac{1}{2},\ 20 + 1.96\sqrt{\frac{20(32-20)}{32}} - \frac{1}{2} \right]$$
$$\approx [15.13, 24.87] \tag{4.58}$$

The conditions (4.56) hold also for k_2, so

$$\text{CI}(\omega_2, 95) = \left[12 - 1.96\sqrt{\frac{12 \times 20}{32}} + \frac{1}{2},\ 12 + 1.96\sqrt{\frac{12 \times 20}{32}} - \frac{1}{2} \right]$$
$$\approx [7.13, 16.87] \tag{4.59}$$

There is a slight overlap of the 95 percent CIs as shown in Figure 4.2, therefore WER will label this cell according to the "least wrong" classifier regardless of the counts ($k_1 = 20$ and $k_2 = 12$) in the cell.

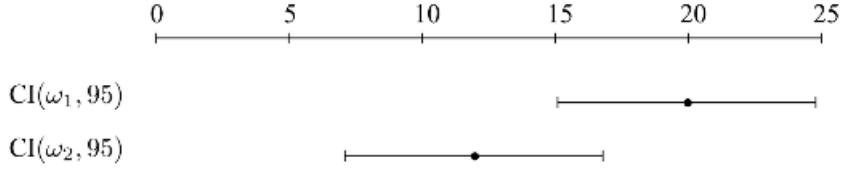


Fig. 4.2 *CI overlap.*

Example: The "Least Wrong" Classifier. In the example with the overlapping CIs, the label for the cell is produced by the least wrong among the three classifiers. To identify this classifier, the confusion matrices are used. Let the three confusion matrices be as follows

$$CM^1 \qquad CM^2 \qquad CM^3$$

$$\begin{bmatrix} 60 & 27 \\ 42 & 71 \end{bmatrix} \quad \begin{bmatrix} 53 & 34 \\ 14 & 99 \end{bmatrix} \quad \begin{bmatrix} 70 & 17 \\ 33 & 80 \end{bmatrix}$$

where, again, $cm^i_{j,k}$ is the number of objects with true class label ω_j, classified as ω_k by D_i. Then the least wrong classifier is found by

$$\hat{P}(error \text{ and } s_1 = \omega_2) = \hat{P}(\omega_1 | s_1 = \omega_2)\hat{P}(s_1 = \omega_2) = \frac{27}{98} \cdot \frac{98}{200} = \frac{27}{200}$$

$$\hat{P}(error \text{ and } s_2 = \omega_1) = \hat{P}(\omega_2 | s_2 = \omega_1)\hat{P}(s_2 = \omega_1) = \frac{14}{67} \cdot \frac{67}{200} = \frac{14}{200}$$

$$\hat{P}(error \text{ and } s_3 = \omega_2) = \hat{P}(\omega_1 | s_3 = \omega_2)\hat{P}(s_3 = \omega_2) = \frac{17}{97} \cdot \frac{97}{200} = \frac{17}{200}$$

As $\hat{P}(error \text{ and } s_2 = \omega_1) = \frac{14}{200}$ is the smallest of the three, classifier D_2 is authorized to label $\mathbf{x}$, and thus the assigned class is ω_1. Note that even though the majority disagree with this decision, D_2's decision is taken by the authoritative power of the least wrong classifier.

The Normal approximation of the Binomial distribution is convenient but the confidence intervals are sometimes too wide because the training sample is not large enough. Besides, for many cells in the table, there will be just a few examples, and then the rules of thumb (4.56) might not hold. Then another estimate of the CI is needed, and the Chebyshev inequality is suggested for this purpose [143]. However, the CIs using the Chebyshev inequality are even wider, which will easily lead to abandoning the multinomial model completely and resorting to the least wrong classifier decision for all values of $\mathbf{s}$. To prevent such a situation and still make use of the CIs, we can decide to consider, say, 70 percent CIs instead of 95 percent CIs. During the design, we can find out what portion of the cells are labeled by the multinomial model and what are labeled by the least wrong classifier, and subsequently pick the percentage of confidence so as to optimize the balance according to the designer's insight.

4.6 PROBABILISTIC APPROXIMATION

Consider $\mathbf{s} = [s_1, \ldots, s_L]^T$, $s_j \in \Omega$ to be an L-dimensional random variable consisting of the class label outputs. An approximation of $P(\mathbf{x}|\omega_k)$ is obtained based on

first-order dependencies as explained below. These estimates are used to calculate the support for the classes

$$\mu_j(\mathbf{x}) = P(\omega_j)P(\mathbf{s}|\omega_j), \qquad j = 1, \ldots, c \tag{4.60}$$

From probability theory

$$\begin{aligned} P(\mathbf{s}|\omega_k) = P(s_{m_1}|\omega_k)P(s_{m_2}|s_{m_1}, \omega_k)P(s_{m_3}|s_{m_1}, s_{m_2}, \omega_k) \\ \ldots P(s_{m_L}|s_{m_1}, \ldots, s_{m_{L-1}}, \omega_k) \end{aligned} \tag{4.61}$$

for any permutation $(m_1, \ldots, m_L)$ of the integers $1, 2, \ldots, L$. The simplest implementation of Eq. (4.61) requires conditional independence (the naive Bayes approximation) so that

$$P(\mathbf{s}|\omega_k) = \prod_{i=1}^{L} P(s_i|\omega_k), \qquad j = 1, \ldots, c \tag{4.62}$$

Although the naive Bayes method has been shown to be practically robust enough [140,141], further studies have been carried out aiming at a more accurate estimation of $P(\mathbf{s}|\omega_j)$. Kang et al. [144–146] suggest using approximations of kth order for combining classifiers. In other words, the probabilities in the product (4.61) can be conditioned by at most k variables.

Below we explain an algorithm due to Chow and Liu [147] for approximating a joint probability distribution by a product of first-order distributions. To simplify notation we shall consider $P(\mathbf{s})$ bearing in mind that the same algorithm will be used separately for each $P(\mathbf{s}|\omega_k)$, $k = 1, \ldots, c$. Each of the outputs s_{m_i} is paired with another output indexed by $m_{j(i)}$ where $0 \le j(i) \le i$. A fictional output s_0 is introduced that is used to allow for the following relationship

$$P(s_i|s_0) = P(s_i)$$

Each of the probabilities participating in the product, $P(s_{m_i}|s_{m_{j(i)}})$, is conditioned on a single other output encountered already. For example, suppose that the chosen permutation of indices is $1, 2, \ldots, L$. Then s_4 can only be conditioned by s_0, s_1, s_2, or s_3. The suggested approximation is

$$P(\mathbf{s}) = \prod_{i=1}^{L} P(s_{m_i}|s_{m_{j(i)}}), \qquad 0 \le j \le i \tag{4.63}$$

where $(m_1, \ldots, m_L)$ is unknown permutations of integers $1, 2, \ldots, L$. Equation (4.63) is called a probability distribution of first-order tree dependence. A distribution of type (4.63) can be conveniently visualized using a line diagram as shown in Figure 4.3.

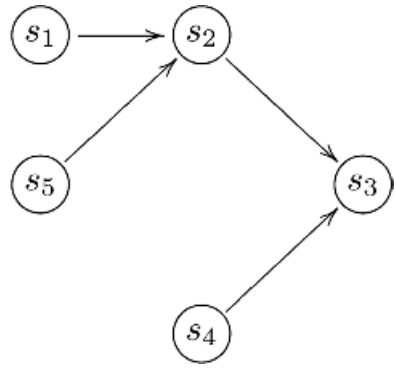


Fig. 4.3 *A diagram of a first-order dependence tree corresponding to the probability approximation* $P(\mathbf{s}) = P(s_3)P(s_2|s_3)P(s_4|s_3)P(s_1|s_2)P(s_5|s_2)$. *The permutation* $m_1, \ldots, m_L$ *is* 3, 2, 4, 1.5.

Each variable is represented as a node in the diagram. Two variables s_i and s_k are joined by an arrow from s_i to s_k if $k = j(i)$. Thus, as $P(s_2|s_3)$ participates in the product, there is an arrow from s_2 to s_3. For all $P(s_i|s_0)$, the node s_i will appear on the diagram but will have no successors (this will indicate a term $P(s_i)$ in the product (4.63)).

Chow and Liu use an information theoretic measure to assess the goodness of the approximation of the true probability P by $\hat{P}$

$$\phi(P, \hat{P}) = \sum_{\mathbf{s}} P(\mathbf{s}) \log \frac{P(\mathbf{s})}{\hat{P}(\mathbf{s})} \tag{4.64}$$

For an ideal approximation where $P(\mathbf{s}) \equiv \hat{P}(\mathbf{s})$ for any $\mathbf{s}$, $\phi(P, \hat{P}) = 0$. Otherwise, $\phi(P, \hat{P}) > 0$. The closer $\hat{P}$ and P are, the lower the value of the measure. The approximation of the L-dimensional joint distribution by a distribution of first-order tree dependence derived in Ref. [147] is optimal in terms of $\phi(P, \hat{P})$. Let T_L be the set of all possible first-order dependence trees. We seek to find a tree $\tau \in T_L$ such that the approximation $\hat{P}_\tau$ of the true distribution P satisfies

$$\phi(P, \hat{P}_\tau) \leq \phi(P, \hat{P}_t), \qquad \forall t \in T_L \tag{4.65}$$

An exhaustive search among the trees in T_L might be infeasible for large L, as there are $L^{(L-2)}$ possible trees with L nodes. Therefore, an optimization procedure for minimizing $\phi(P, \hat{P}_t)$ is devised. It is based on the mutual information between two variables s_i and s_k

$$I(s_i, s_k) = \sum_{s_i, s_k} P(s_i, s_k) \log \frac{P(s_i, s_k)}{P(s_i)P(s_k)} \tag{4.66}$$

The mutual information is nonnegative. It takes value zero only if the two variables are statistically independent. We can attach weights to each edge of the dependence tree illustrating the first-order dependencies. The weight of a branch from s_i to s_k will be $I(s_i, s_k)$. To measure the quality of a dependence tree, we sum up the weights of all its branches and call the sum *the weight of the tree*. Intuitively, the tree captures most information if it has the highest possible weight among the trees in T_L.

4.6.1 Calculation of the Probability Estimates

Example: Estimation of the First-Order Probabilities. Let $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$. Let the data set $\mathbf{Z}$ consist of 1000 data points, 100 from ω_1, 200 from ω_2, 300 from ω_3, and 400 from ω_4. Five classifiers were trained on $\mathbf{Z}$ and the coincidence matrices between all pair of classifiers were stored, separately for each class.

A possible coincidence matrix produced by classifiers D_2 and D_3 with regard to class ω_1, that is, using only the 100 elements from this class in $\mathbf{Z}$, is

		$s_3 = D_3(\mathbf{x})$			
		ω_1	ω_2	ω_3	ω_4
$s_2 = D_2(\mathbf{x})$	ω_1	43	7	10	4
	ω_2	10	2	1	1
	ω_3	6	2	1	2
	ω_4	8	0	1	2

Assume that the tree found through the analysis is the one shown in Figure 4.3. A relevant coincidence matrix would be that of D_2 and D_3 as $P(s_2|s_3)$ participates in the expression. Note that in this example we are estimating $P(\mathbf{s}|\omega_1)$, so all terms in the approximation are conditioned also by ω_1; that is,

$$P(\mathbf{s}|\omega_1) = P(s_3|\omega_1)P(s_2|s_3, \omega_1)P(s_4|s_3, \omega_1)P(s_1|s_2, \omega_1)P(s_5|s_2, \omega_1) \tag{4.67}$$

The estimate $\hat{P}(s_2|s_3, \omega_1)$ is calculated from the coincidence matrix of D_2 and D_3 for class ω_1. For example, let $s_2 = \omega_3$ and $s_3 = \omega_1$,

$$\begin{aligned} \hat{P}(s_2|s_3, \omega_1) &= \hat{P}(s_2 = \omega_3 \mid s_3 = \omega_1, (\text{true class} =)\omega_1) \\ &= \frac{6}{43 + 10 + 6 + 8} = \frac{6}{67} \end{aligned} \tag{4.68}$$

The value for the multipliers in the approximation that are only conditioned by the class is taken from the confusion matrix of the respective classifier. In this

example, $P(s_3|\omega_1)$ is estimated as the proportion labeled as s_3 by D_3 out of the 100 elements from $\mathbf{Z}$ from ω_1. For $s_3 = \omega_1$,

$$\hat{P}(s_3 = \omega_1|\omega_1) = \frac{43 + 10 + 6 + 8}{100} = \frac{67}{100} \tag{4.69}$$

The final class label is inferred using the Bayes formula. It is sufficient to calculate $P(\omega_i)P(\mathbf{s}|\omega_i)$ for $i = 1, \ldots, 4$, and decide in favor of the class that maximizes the expression.

To understand the gain in the representation of the combiner compared to the BKS table, we can estimate the necessary memory space for each of the combiners. For the BKS lookup table, we need c^L table cells. For the first-order approximation, for each of the c classes, we will need at most $L - 1$ coincidence matrices and one set of c values for the term conditioned only on the (true) class label ($P(s_3|\omega_1)$ in the example above). This amounts to $c \times [(L - 1) \times c^2 + L]$. Thus, for the naive Bayes combiner, the required space is $L \times c^2$. Finally, for the above example:

- BKS requires $c^L = 4^5 = 1024$ cells;
- The first-order probabilistic combination requires $c \times [(L - 1) \times c^2 + L] = 4 \times [(5 - 1) \times 4^2 + 5] = 276$ cells;
- Naive Bayes requires $L \times c^2 = 5 \times 4^2 = 80$ cells; and
- Majority vote requires none.

For small-scale examples such as this, there will be no substantial difference. However, when character recognition is concerned with number of classes $c = 26$, multinomial methods might become infeasible and first-order approximation might be just the solution. Of course the cost of implementing the combiner depends also on the complexity of the algorithm used.

4.6.2 Construction of the Tree

The dependence tree for each class is constructed using the subset of data set $\mathbf{Z}$ for that class. According to the proof given in Appendix 4B, the tree has to be constructed to maximize the total mutual information; that is,

$$\max \sum_{\{\text{all branches}\}} I(s_i, s_k) \tag{4.70}$$

The algorithm for constructing the tree is as follows. After calculating the pairwise mutual information for all pairs (s_i, s_k), $1 \le i, k \le L$, $i \neq k$, we make use of the *minimum spanning tree* (MST) algorithm. This MST produces an undirected graph. To recover the terms in the approximation we convert it into a *directed* maximum mutual information tree.

Construction of a probabilistic tree for class ω_j

1. Input the matrix S of size $N_j \times L$ containing the label outputs of the L classifiers for the N_j elements of $\mathbf{Z}$ from class ω_j.
2. Calculate an $L \times L$ matrix MI containing the pairwise mutual information between every pair of classifier outputs, s_i and s_k.
3. Since we want the tree with the largest mutual information, use $-MI$ as a distance matrix and run the minimum spanning tree (MST) algorithm (see the single linkage algorithm in Figure 1.14).
4. Construct a directed tree from the MST using the following steps.
   4.1. Start with an empty set NewTree. Suppose that the branches of the MST are arranged in order of decreasing mutual information. Take the first branch, say from s_1 to s_2, mark both its ends as "used", and add to NewTree a directed branch from s_1 to s_2.
   4.2. Take the next branch in the MST list, say from s_i to s_k. If s_i is marked as "used", append a branch from s_i to s_k to NewTree and mark s_k as "used". If s_k is marked as "used", append to NewTree a branch from s_k to s_i and mark s_i as "used". If none of the two is marked then append the branch in any order and mark both ends as "used".
   4.3. Continue from 4.2 until all branches of MST are added to NewTree.

Fig. 4.4a *Construction of a first-order probability dependence tree for class ω_j.*

One possible algorithm is described in Figure 4.4*a*. The tree is constructed by subsequently adding branches, one at a time. Figure 4.4*b* shows an example of constructing a directed tree from a minimum spanning tree (MST).

The calculation of the mutual information is straightforward as illustrated in the example below.

Example: Calculation of the Mutual Information. Let us calculate $I(s_2, s_3)$ using the coincidence matrix from the previous example. From Eq. (4.66)

$$\hat{I}(s_2, s_3) = \sum_{\text{all cells}} \frac{\text{cell entry}}{100} \log\left(\frac{\text{cell entry} \times 100}{[\text{column sum}] \times [\text{row sum}]}\right) \tag{4.71}$$

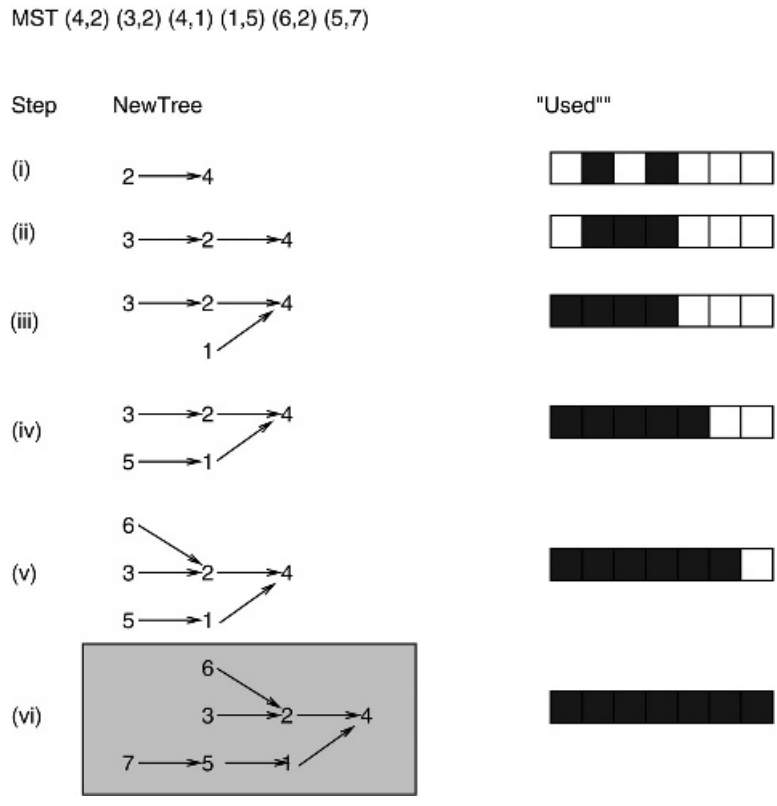


Fig. 4.4b *Illustration of finding a directed tree from a minimum spanning tree (MST). The probability approximation is* $P(\mathbf{s}|\omega_k) = P(s_4|\omega_k)P(s_2|s_4, \omega_k)\cdots P(s_3|s_2, \omega_k)P(s_1|s_4, \omega_k)P(s_5|s_1, \omega_k)P(s_6|s_2, \omega_k)P(s_7|s_5, \omega_k)$.

Thus for our example, excluding all cells with zero entries

$$\hat{I}(s_2, s_3) = \frac{43}{100}\log\left(\frac{43 \times 100}{[43+10+6+8] \times [43+7+10+4]}\right) + \cdots + \frac{2}{100}\log\left(\frac{2 \times 100}{[4+1+2+2] \times [8+0+1+2]}\right) \approx 0.0332 \tag{4.72}$$

The next example shows the results from the algorithm in Figure 4.4*a*. It should be noted that there is more than one way to write the approximation formula for the same dependence tree. This is due to the symmetry of the mutual information: $I(s_i, s_k) = I(s_k, s_i)$.

TABLE 4.8 A Hypothetical Distribution of the Classifier Outputs for $D_1, \ldots, D_5$ and Two Classes. To Save Space, Only the Class Subscripts Are Shown (1 and 2, Instead of ω_1 and ω_2).

Outputs	Freq.	Outputs	Freq.	Outputs	Freq.	Outputs	Freq.
11111	5	12111	8	21111	7	22111	4
11112	4	12112	7	21112	10	22112	9
11121	10	12121	10	21121	9	22121	8
11122	8	12122	9	21122	8	22122	15
11211	3	12211	10	21211	5	22211	8
11212	14	12212	14	21212	10	22212	11
11221	10	12221	12	21221	7	22221	16
11222	10	12222	12	21222	8	22222	19

Example: Results from the Tree Construction Algorithm. Table 4.8 shows a hypothetical distribution of five classifier outputs across a data set of 300 data points. Assume that all the data comes from class ω_1. Using the algorithm in Figure 4.4, we build a first-order dependence tree to approximate the probability $P(\mathbf{s}|\omega_1)$.

The algorithm produced the following four pairs of classifiers: (4,5), (5,2), (5,1), and (2,3). One possible directed tree is shown in Figure 4.5 together with the approximation formula for $P(\mathbf{s}|\omega_1)$.

After deriving all the approximations for all the classes, $P(\mathbf{s}|\omega_k)$, $k = 1, \ldots, c$, the operation of the combiner is illustrated by the following example.

Example: Combining Classifiers by First-Order Dependence Trees – Operation. To classify an $\mathbf{x} \in \Re^n$, we first obtain the label vector $\mathbf{s} = [s_1, s_2, \ldots, s_L]^T$. Next we show how to calculate $P(\mathbf{s}|\omega_1)$.

We assume again the 300 data points whose vote distribution is given in Table 4.8 come from class ω_1. The first-order dependence tree in Figure 4.5 shows the relevant pairwise dependencies: (D_4, D_5), (D_5, D_2), (D_5, D_1), and (D_2, D_3). Also, we have to use the confusion matrix of D_4. Using the data in Table 4.8, the pairwise dependencies are found to be as shown in Table 4.9. A hypothetical confusion matrix for classifier D_4 is displayed in Table 4.10.

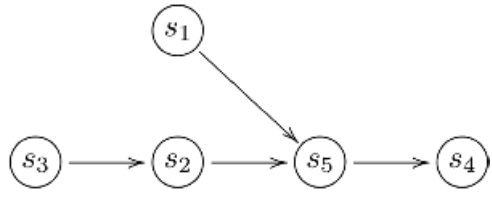


Fig. 4.5 The dependence tree corresponding to the data in Table 4.8. The probability approximation is $P(\mathbf{s}|\omega_1) = P(s_4|\omega_1)P(s_5|s_4, \omega_1)P(s_1|s_5, \omega_1)P(s_2|s_5, \omega_1)P(s_3|s_2, \omega_1)$.

TABLE 4.9 Relevant Pairwise Coincidence Matrices for Approximating $P(\mathbf{s}|\omega_1)$ for the Dependence Tree in Figure 4.5.

	D_5	
D_4	1	2
1	50	79
2	82	89

	D_1	
D_5	1	2
1	68	64
2	78	90

	D_2	
D_5	1	2
1	56	76
2	72	96

	D_3	
D_2	1	2
1	61	67
2	70	102

Let $\mathbf{s} = [s_1, \ldots, s_5] = [\omega_1, \omega_1, \omega_2, \omega_1, \omega_2]^T$, or, in short notation, 11212. The approximation of $P(\mathbf{s}|\omega_1)$ is calculated as

$$P(s_4 = \omega_1|\omega_1) = \frac{129}{300} \text{ (from the confusion matrix in Table 4.10)}$$

$$P(s_5 = \omega_2|s_4 = \omega_1, \omega_1) = \frac{79}{50 + 79}$$

$$P(s_1 = \omega_1|s_5 = \omega_2, \omega_1) = \frac{78}{78 + 90}$$

$$P(s_2 = \omega_1|s_5 = \omega_2, \omega_1) = \frac{72}{72 + 96} \tag{4.73}$$

$$P(s_3 = \omega_2|s_2 = \omega_1, \omega_1) = \frac{67}{61 + 67}$$

$$P(\mathbf{s}|\omega_1) = \frac{129}{300} \cdot \frac{79}{129} \cdot \frac{78}{168} \cdot \frac{72}{168} \cdot \frac{67}{128} \approx 0.0274 \tag{4.74}$$

TABLE 4.10 The Hypothetical Confusion Matrix for Classifier D_4.

	Guessed Label	
True Label	1	2
1	129	171
2	12	88

To label $\mathbf{x}$, we need the discriminant scores $P(\omega_1)P(\mathbf{s}|\omega_1)$ and $P(\omega_2)P(\mathbf{s}|\omega_2)$. From the above calculations, and from the confusion matrix for D_4, we can only estimate $P(\omega_1)P(\mathbf{s}|\omega_1) \approx 0.75 \times 0.0274 = 0.02055$. For the score for class ω_2, the approximation of $P(\mathbf{s}|\omega_2)$ has to be obtained, using the respective dependence tree. The highest score will determine the label of $\mathbf{x}$.

4.7 CLASSIFIER COMBINATION USING SINGULAR VALUE DECOMPOSITION

Using *correspondence analysis*, Merz [148] proposes to map the space spanned by the classifier outputs to a low-dimensional real space. Each class label is represented by a prototype in the new space. The classification decision is made by the nearest prototype. Finding a mapping T is in fact *feature extraction* in the space Ω^L spanned by the classifier outputs $\mathbf{s} = [s_1, \ldots, s_L]$, called the *intermediate feature space*.

The procedure outlined below follows the idea of Ref. [148] except for some details as explained in the text.

Consider a matrix $M = \{m_{i,j}\}$ of size $N \times (L \cdot c)$ containing the outputs of the ensemble for the N data points in $\mathbf{Z}$. Each classifier output (a class label) is represented as a binary vector with 1 at the position of the suggested class label, and 0s at the other $c - 1$ positions. A row in M consists of the concatenated outputs of the L classifiers for the respective data point. For example, for $L = 3$ classifiers, $c = 4$ classes, and outputs $\mathbf{s} = [\omega_4, \omega_2, \omega_1]^T$, the row in M for this object will be (0,0,0,1,0,1,0,0,1,0,0,0).

Correspondence analysis[20] is a technique to analyze two-way and multiway tables containing some measure of correspondence between rows and columns. It is similar to factor analysis but is designed for categorical variables. In our case, the table of interest is the matrix M. We may look at M as a description of N objects in an $L \cdot c$-dimensional binary space. The challenge is to "equivalently" represent the N objects in a lower-dimensional space, for example, two-dimensional, for visualization purposes. We can then build a classifier on this new representation. The classifier model suggested in Ref. [148] is the Nearest Mean. It is impossible to predict whether or not the classifier built in the new feature space will be better than a classifier built on the $L \cdot c$-dimensional binary data in M. There is no immediate gain in transforming the data set and disregarding in the process any information that would help discriminating between the classes.[21] Merz suggests to augment M by c more columns encoding the true class label of each object [148], thus taking into account some discrimination information. For example, assume that the true label of $\mathbf{z}_j$ is ω_3. In a four-class problem, the addition to the row of M corresponding to $\mathbf{z}_j$ will be 0010. If the number of classifiers is large, the additional c columns are not going to make a significant difference to the representation in the new low-

[20] See *http://www.statsoft.com/textbook/stcoran.html.*

[21] Similarly, principal component analysis (PCA) and Karhunen–Loéve expansion used in pattern recognition to capture the relationship between the objects do not necessarily facilitate the ensuing classification.

dimensional space. Besides, when a new object comes for classification, we will have to *guess* its label (the additional c columns) before plotting it into the new space. Merz uses all c label guesses, one at a time, to see with which guessed label the representation of $\mathbf{x}$ is closest to a class prototype in the new space [148].

Below we detail the algorithm for finding the new space with a lower dimension $\delta \leq \min\{N, (L \cdot c)\}$ for our (nonaugmented) binary matrix M.

First, normalize M so that the sum of all of its elements is 1.

$$\mathcal{M} = \frac{1}{\sum_{i=1}^{N} \sum_{j=1}^{L \cdot c} m_{i,j}} M = \frac{1}{(N \cdot L)} M \tag{4.75}$$

Denote by c_j the *column mass* of column j in $\mathcal{M}$; that is,

$$c_j = \frac{1}{N \cdot L} \sum_{i=1}^{N} m_{i,j} \tag{4.76}$$

Let $\mathbf{c} = [c_1, \ldots, c_{L \cdot c}]^T$ be the vector with column masses, and $\mathbf{r} = [(1/N), \ldots, (1/N)]^T$ be the vector with row masses (of size $(N \times 1)$). Use the two vectors to form the the following diagonal matrices

$$\begin{aligned} M_c &= \text{diag}\left\{\frac{1}{\sqrt{c_1}}, \ldots, \frac{1}{\sqrt{c_{L \cdot c}}}\right\}_{((L \cdot c) \times (L \cdot c))} \\ M_r &= \text{diag}\left\{\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}}\right\}_{(N \times N)} \end{aligned} \tag{4.77}$$

Calculate the standardized residual matrix

$$A = M_r(\mathcal{M} - \mathbf{r}\mathbf{c}^T)M_c \tag{4.78}$$

The *singular value decomposition* (SVD) of A is then found as $A = U\Gamma V^T$, and the principal coordinates of the N rows and the $L \cdot c$ columns are calculated as F and G respectively

$$\begin{aligned} F &= M_r \cdot U \cdot \Gamma \text{ (rows)} \\ G &= M_c \cdot V \cdot \Gamma^T \text{ (columns)} \end{aligned} \tag{4.79}$$

At this stage we can plot the data set $\mathbf{Z}$ on the plane spanned by the first two components of F, and show the class labels by different markers. Figure 4.6 depicts the results from the run of the above algorithm on the *Pima Indian Diabetes* data set from the UCI Machine Learning Repository. Fifteen classifiers were trained on 90 percent of the data set, that is, $N = 691$, $L = 15$, $c = 2$, and so M is of size

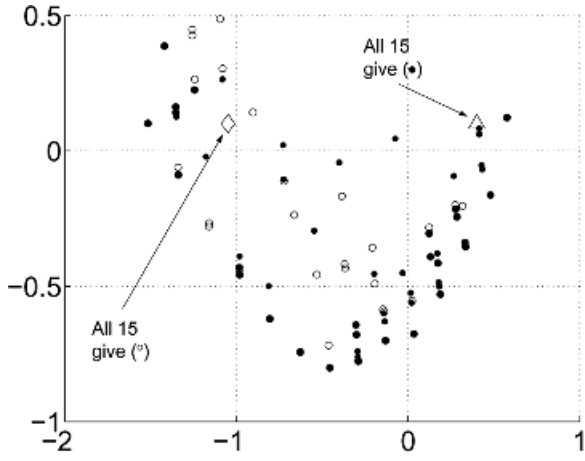


Fig. 4.6 *Scatterplot of the 691 points in the Pima training set (a randomly chosen 90 percent of the data) in the plane spanned by the first two principal axes found through the first SVD method: only the 15 classifier outputs are concatenated.*

691×30. The number of dimensions will be at most as the number of nonzero singular values of A. Not all c columns per class are actually needed because from the $c-1$ binary values, we can always derive the cth one. Then the independent columns of M will be at most $L \times (c-1)$, so, in this example, there could be a maximum of 15 dimensions. The first dimension corresponds to the highest singular value and "explains" most of the relationship among the objects, and the other dimensions follow in descending order of importance. Each of the 15 classifiers is a Multi-Layer Perceptron neural network (MLP NN) with one hidden layer containing five nodes. Backpropagation applied for 100 epochs was used for training (Matlab Toolbox).

As Figure 4.6 shows, the new space hardly helps in solving the original classification problem. To argue our point that appending the true class label will not make much difference, Figure 4.7 shows the results of the same experiment but with the true class labels added as the last two columns of M. Observing Figure 4.7, there is no evidence that the SVD improves the chances for a better combination when this more elaborate procedure is employed. The plots can only give us an idea about the difficulty of the problem at hand, so the true value of the SVD method lies with illustration rather than improved accuracy of the ensemble.

For completeness, we include the second part of the algorithm, which explains the classification stage. The dimensionality δ has to be decided next. The restriction is that $\delta \leq n_0 \leq \min\{N, (L \cdot c)\}$, where n_0 is the total number of positive singular values of A. Then a classifier is trained on F, with all N rows but with only the first δ columns. The class labels for the N objects are the labels of the $\mathbf{z}_j$'s. For example, if we use a linear discriminant classifier in the new space, there will be $c(\delta+1)$ parameters needed for the classification of an object represented by its δ values. To convert a label vector $\mathbf{s}$ to a representation by δ values, we first find

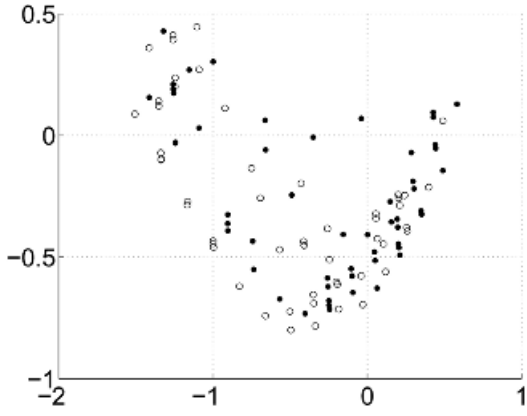


Fig. 4.7 *Scatterplot of the 691 points in the Pima training set (a randomly chosen 90 percent of the data) in the plane spanned by the first two principal axes found through second SVD method: concatenated are the 15 classifier outputs plus the true labels.*

the concatenated binary representation of $\mathbf{s}$, for example, $\mathbf{s}_b$, and then calculate

$$\mathbf{s}_\delta^T = \frac{1}{L}\mathbf{s}_b^T \cdot G_r \cdot \Gamma_r \tag{4.80}$$

where the "r" in G_r and Γ_r stands for "reduced." Only the first δ columns of G are taken as G_r, and a diagonal matrix is "cut" from Γ, of size $\delta \times \delta$, containing as its leading diagonal the δ highest singular values of A in descending order. We can fuse the constant terms, thereby forming a single transformation matrix $T = (1/L) \cdot G_r \cdot \Gamma_r$ of size $(L \cdot c) \times \delta$.

Example: Singular Value Decomposition Classification of an Unknown x. For the example shown in Figure 4.6, the number of nonzero singular values of A is 14. Pick δ to be 4. Then G_r will have 30 rows and 4 columns, and Γ_r will be of size 4×4:

$$\Gamma_r = \begin{bmatrix} 0.83 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0.20 & 0 \\ 0 & 0 & 0 & 0.13 \end{bmatrix} \tag{4.81}$$

The transformation matrix T is calculated as explained above. Assume that all classifiers gave the same class label ω_1 for $\mathbf{x}$. Then

$$\mathbf{s}_b = [10101010101010101010101010101010]^T$$

Next we use Eq. (4.80) to find $\mathbf{s}_\delta = [0.40, 0.01, 0.00, 0.00]^T$. Similarly, we can derive the $\mathbf{s}_\delta$ for an $\mathbf{x}$ labeled by all 15 classifiers as ω_2. In this case, $\mathbf{s}_\delta = [-1.05, 0.01, 0.01, 0.00]^T$. The two prototypes calculated in this way are plotted in Figure 4.6.

4.8 CONCLUSIONS

This chapter describes various methods for combining class label outputs. Formally speaking, we transformed the problem to find a mapping

$$D : \Re^n \to \Omega \tag{4.82}$$

into a problem of finding a mapping

$$\mathcal{F} : \Omega^L \to \Omega \tag{4.83}$$

We can call Ω^L an *intermediate feature space*, which in this case is discrete. The methods described here varied on the assumptions and also on the complexity of their implementation. Table 4.11 summarizes the methods in this chapter.

Here we bring the results from a single illustrative experiment. The Pima Indian Diabetes data set was used again, in a 10-fold cross-validation experiment (rotated 90 percent for training and 10 percent for testing). Figure 4.8 shows the mean accuracies from the 10 runs, for training and for testing. The single best classifier in the ensemble is also displayed to set up a baseline for the ensemble accuracy.

The individual classifiers were MLP NNs, each one with a single hidden layer containing 25 nodes, trained starting with different random initializations. Nine classifiers were trained on each training/testing split of the data set. The standard back-

TABLE 4.11 A Summary of the Label Fusion Methods.

Method	Assumptions (for Optimality)	Memory Requirements (in Number of Parameters Needed for Its Operation)
Majority vote	None	None
Weighted majority vote	None	L
Naive Bayes	Conditional independence	$L \times c^2$
Behavior knowledge space (BKS)	None	c^L
Wernecke	None	c^L
First-order dependence tree	Conditional first-order dependence	$c \times [(L-1) \times c^2 + L]$
SVD combination	None	$[L \times c + (c+1)]\delta$ $(\delta \leq \min\{N, (L \cdot c)\})$

SVD, singular value decomposition.

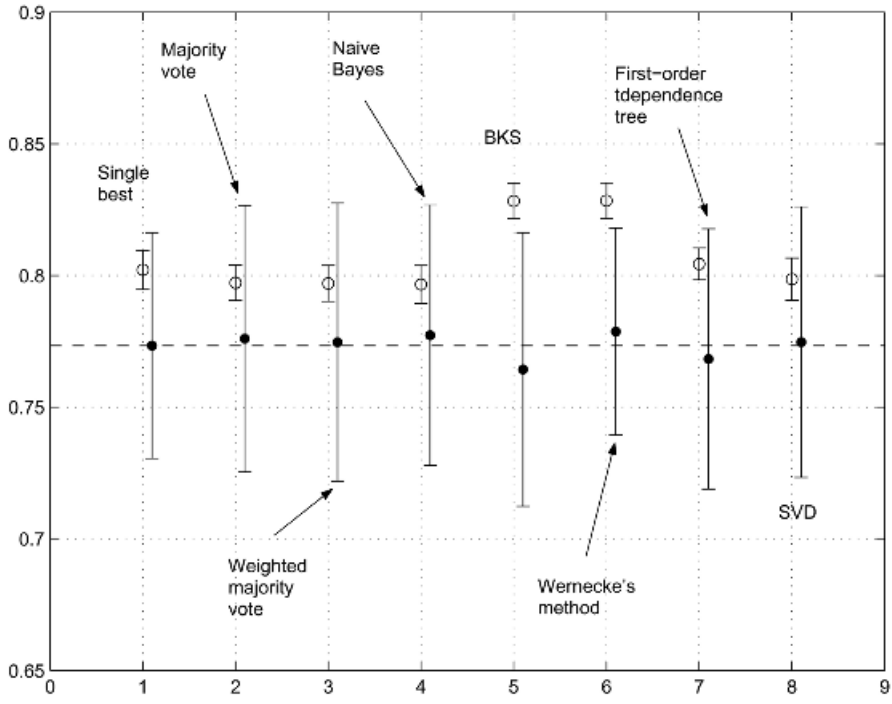


Fig. 4.8 *Results from a 10-fold cross-validation with the Pima Indian Diabetes database from UCI. The combination methods discussed in this chapter are contrasted against each other and also against the single best classifier in the team of nine classifiers. Each individual classifier is a multi-layer perceptron (MLP) with one hidden layer containing 25 nodes. The training and testing results for each method are shown next to each other, with the mean value (○ for the training, and ● for the testing) and one standard deviation on each side, calculated from the 10 cross-validation runs.*

propagation was used from the Matlab Neural Network Toolbox. Each network was trained for 300 epochs. Figure 4.8 reveals that

1. There is hardly any improvement over the single best member of the team. (The dashed line sets up the standard for the testing accuracy.) The single best classifier was selected based on the *training* accuracies of the ensemble members. This means that there could have been even higher single accuracies among the testing results. The lack of improvement could be attributed to the lack of variability (diversity) in the ensemble.[22]
2. As expected, the training accuracy is higher than the testing accuracy, and has much smaller variability. A notable overtraining occurs with the BKS and Wernecke's method. However, while BKS is the worst in this illustration, Wernecke's method is one of the best contestants, despite the overtraining.

[22] Diversity in classifier ensembles will be discussed at length in Chapter 10.

3. None of the methods shows a clear dominance above the remaining methods, which again confirms the "no panacea" principle in pattern recognition.

APPENDIX 4A MATAN'S PROOF FOR THE LIMITS ON THE MAJORITY VOTE ACCURACY

Here we give a sketch of the proof as offered in Ref. [137].

Theorem. Given is a classifier ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$. Suppose that classifiers D_i have accuracies p_i, $i = 1, \ldots, L$, and are arranged so that $p_1 \leq p_2 \cdots \leq p_L$. Let $k = l + 1 = (L + 1)/2$. Then

1. The upper bound of the majority vote accuracy of the ensemble is

$$\max P_{\text{maj}} = \min \{1, \Sigma(k), \Sigma(k-1), \ldots, \Sigma(1)\}, \tag{A.1}$$

   where

$$\Sigma(m) = \frac{1}{m} \sum_{i=1}^{L-k+m} p_i, \qquad m = 1, \ldots, k \tag{A.2}$$

2. The lower bound of the majority vote accuracy of the ensemble is

$$\min P_{\text{maj}} = \max \{0, \xi(k), \xi(k-1), \ldots, \xi(1)\} \tag{A.3}$$

   where

$$\xi(m) = \frac{1}{m} \sum_{i=k-m+1}^{L} p_i - \frac{L-k}{m}, \qquad m = 1, \ldots, k \tag{A.4}$$

Proof. (sketch)

Upper Bound. The accuracy p_i is the average accuracy of D_i across the whole feature space $\Re^n$, and can be written as

$$p_i = \int_{\Re^n} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{A.5}$$

where $\mathcal{I}_i$ is an indicator function for classifier D_i, defined as

$$\mathcal{I}_i(\mathbf{x}) = \begin{cases} 1, & \text{if } D_i \text{ recognizes correctly } \mathbf{x}, \\ 0, & \text{otherwise,} \end{cases} \tag{A.6}$$

and $p(\mathbf{x})$ is the probability density function of $\mathbf{x}$. The majority vote accuracy is the probability of having k or more correct votes, averaged over the feature space $\Re^n$.

$$P_{\text{maj}} = \int_{\sum \mathcal{I}_i(\mathbf{x}) \geq k} p(\mathbf{x})\, d\mathbf{x} \tag{A.7}$$

First we note that $P_{\text{maj}} \leq 1$, and then derive a series of inequalities for P_{maj}. For any $\mathbf{x}$ where the majority vote is correct, at least k of the classifiers are correct. Thus

$$\begin{aligned} \sum_{i=1}^{L} p_i &= \sum_{i=1}^{L} \int_{\Re^n} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} = \int_{\Re^n} \sum_{i=1}^{L} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \\ &\geq \int_{\sum \mathcal{I}_i(\mathbf{x}) \geq k} k p(\mathbf{x})\, d\mathbf{x} = k P_{\text{maj}} \end{aligned} \tag{A.8}$$

Then

$$P_{\text{maj}} \leq \frac{1}{k} \sum_{i=1}^{L} p_i \tag{A.9}$$

Let us now remove the most accurate member of the team, D_L, and consider the remaining $L-1$ classifiers

$$\sum_{i=1}^{L-1} p_i = \sum_{i=1}^{L-1} \int_{\Re^n} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{A.10}$$

For each point $\mathbf{x} \in \Re^n$ where the majority vote (using the whole ensemble) has been correct, that is, $\sum \mathcal{I}_i(\mathbf{x}) \geq k$, there are now at least $k-1$ correct individual votes. Thus

$$\begin{aligned} \sum_{i=1}^{L-1} p_i &= \int_{\Re^n} \sum_{i=1}^{L-1} \mathcal{I}_i(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \\ &\geq \int_{\sum_{i=1}^{L} \mathcal{I}_i(\mathbf{x}) \geq k} (k-1) p(\mathbf{x})\, d\mathbf{x} = (k-1) P_{\text{maj}} \end{aligned} \tag{A.11}$$

Then

$$P_{\text{maj}} \leq \frac{1}{k-1} \sum_{i=1}^{L-1} p_i \tag{A.12}$$

Similarly, by dropping from the remaining set the most accurate classifier at a time, we can derive the series of inequalities (A.1). Note that we can remove *any* classifier from the ensemble at a time, not just the most accurate one, and arrive at a similar inequality. Take for example the step where we remove D_L. The choice of the most accurate classifier is dictated by the fact that the remaining ensemble of $L-1$ classifiers will have the smallest sum of the individual accuracies. So as P_{maj} is less than $\frac{1}{k-1}\sum_{i=1}^{L-1} p_i$, it will be less than any other sum involving $L-1$ classifiers that includes p_L and excludes a smaller p_i from the summation.

The next step is to show that the upper bound is achievable. Matan suggests to use induction on both L and k for that [137].

Lower Bound. To calculate the lower bound, Matan proposes to invert the concept, and look again for the upper bound but of $(1 - P_{\text{maj}})$.

APPENDIX 4B PROBABILISTIC APPROXIMATION OF THE JOINT pmf FOR CLASS-LABEL OUTPUTS

The derivation of the algorithm by Chow and Liu [147] is based on two propositions. First, they show that an approximation $\hat{P}_t$ of P based on a dependence tree $t \in T_L$ is optimal if and only if t has the maximum weight among the trees in T_L.

From Eq. (4.63)

$$\begin{aligned}
\log \hat{P}(\mathbf{s}) &= \sum_{i=1}^{L} \log P\big(s_i|s_{j(i)}\big) \\
&= \sum_{i=1}^{L} \log \frac{P(s_i|s_{j(i)})P(s_i)}{P(s_i)} \\
&= \sum_{i=1}^{L} \log\left(\frac{P(s_i|s_{j(i)})}{P(s_i)}\right) + \sum_{i=1}^{L} \log P(s_i) \qquad \text{(B.1)}
\end{aligned}$$

For all $j(i) = 0$, $P(s_i|s_0) = P(s_i)$, and the respective term in the first summation becomes $\log[P(s_i)/P(s_i)] = 0$. For the remaining terms, replacing $P(s_i|s_{j(i)})$ by

$$P(s_i|s_{j(i)}) = \frac{P(s_i, s_{j(i)})}{P(s_{j(i)})}$$

Eq. (B.1) becomes

$$\log \hat{P}(\mathbf{s}) = \sum_{\substack{i=1 \\ j(i)\neq 0}}^{L} \log \frac{P(s_i, s_{j(i)})}{P(s_i)P(s_{j(i)})} + \sum_{i=1}^{L} \log P(s_i) \qquad \text{(B.2)}$$

Then from Eq. (4.64)

$$\phi(P, \hat{P}_t) = -\sum_{\mathbf{s}} P(\mathbf{s}) \log \hat{P}(\mathbf{s}) + \sum_{\mathbf{s}} P(\mathbf{s}) \log P(\mathbf{s})$$

$$= -\sum_{\mathbf{s}} P(\mathbf{s}) \sum_{i=1, j(i)\neq 0}^{L} \log \frac{P(s_i, s_{j(i)})}{P(s_i)P(s_{j(i)})}$$

$$- \sum_{\mathbf{s}} P(\mathbf{s}) \sum_{i=1}^{L} \log P(s_i) + \sum_{\mathbf{s}} P(\mathbf{s}) \log P(\mathbf{s}) \tag{B.3}$$

Since s_i is a component of $\mathbf{s}$, $P(s_i)$ is the marginal probability given by

$$P(s_i) = \sum_{s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_L} P(s_1, \ldots, s_L)$$

Then, as $\log P(s_i)$ is a constant for all summations except the one on s_i,

$$\sum_{\mathbf{s}} P(\mathbf{s}) \log P(s_i) = \sum_{(s_1,\ldots,s_L)} P(s_1, \ldots, s_L) \log P(s_i)$$

$$= \sum_{s_i} \log P(s_i) \sum_{s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_L} P(s_1, \ldots, s_L)$$

$$= \sum_{s_i} P(s_i) \log P(s_i) - H(s_i), \tag{B.4}$$

where $H(s_i)$ is the entropy of the random variable s_i. Also, the entropy of $\mathbf{s}$ is

$$H(\mathbf{s}) = -\sum_{\mathbf{s}} P(\mathbf{s}) \log P(\mathbf{s}) \tag{B.5}$$

Similarly, we can use the marginal pmf for $(s_i, s_{j(i)})$ to arrive at

$$\sum_{\mathbf{s}} P(\mathbf{s}) \sum_{i=1, j(i)\neq 0}^{L} \log \frac{P(s_i, s_{j(i)})}{P(s_i)P(s_{j(i)})} = \sum_{(s_i, s_{j(i)})} P(s_i, s_{j(i)}) \log \frac{P(s_i, s_{j(i)})}{P(s_i)P(s_{j(i)})}$$

$$= I(s_i, s_{j(i)}) \tag{B.6}$$

Substituting Eqs. (B.4), (B.5), and (B.6) into Eq. (B.3), the following expression is obtained:

$$\phi(P, \hat{P}_t) = -\sum_{i=1}^{L} I(s_i, s_{j(i)}) + \sum_{i=1}^{L} H(s_i) - H(\mathbf{s}) \tag{B.7}$$

Since the entropies $H(s_i)$ and $H(\mathbf{s})$ depend only on the true probability distribution and not on the tree, the minimum of $\phi(P, \hat{P}_t)$ (the best approximation) is obtained by maximizing $\sum_{i=1}^{L} I(s_i, s_{j(i)})$, that is, the weight of tree t.

5

Fusion of Continuous-Valued Outputs

The degrees of support for a given input $\mathbf{x}$ can be interpreted in different ways, the two most common being *confidences* in the suggested labels and *estimates of the posterior probabilities* for the classes.

Let $\mathbf{x} \in \Re^n$ be a feature vector and $\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\}$ be the set of class labels. Each classifier D_i in the ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$ outputs c degrees of support. Without loss of generality we can assume that all c degrees are in the interval [0, 1], that is, $D_i : \Re^n \to [0, 1]^c$. Denote by $d_{i,j}(\mathbf{x})$ the support that classifier D_i gives to the hypothesis that $\mathbf{x}$ comes from class ω_j. The larger the support, the more likely the class label ω_j. The L classifier outputs for a particular input $\mathbf{x}$ can be organized in a *decision profile* ($DP(\mathbf{x})$) as the matrix

$$DP(\mathbf{x}) = \begin{bmatrix} d_{1,1}(\mathbf{x}) & \cdots & d_{1,j}(\mathbf{x}) & \cdots & d_{1,c}(\mathbf{x}) \\ d_{i,1}(\mathbf{x}) & \cdots & d_{i,j}(\mathbf{x}) & \cdots & d_{i,c}(\mathbf{x}) \\ d_{L,1}(\mathbf{x}) & \cdots & d_{L,j}(\mathbf{x}) & \cdots & d_{L,c}(\mathbf{x}) \end{bmatrix} \tag{5.1}$$

Output of classifier $D_i(\mathbf{x})$

Support from classifiers $D_1 \cdots D_L$ for class ω_j

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

The methods described in the rest of this chapter use $DP(\mathbf{x})$ to find the overall support for each class and subsequently label the input $\mathbf{x}$ in the class with the largest support. There are two general approaches to this task. First, we can use the fact that the values in column j are the individual supports for class ω_j and derive an overall support value for that class. Denote by $\mu_j(\mathbf{x})$ the overall degree of support for ω_j given by the ensemble. Combination methods that use one column of $DP(\mathbf{x})$ at a time are called in Ref. [149] "class-conscious." Examples from this group are the simple and weighted average, product, and order statistics. Alternatively, we may ignore the context of $DP(\mathbf{x})$ and treat the values $d_{i,j}(\mathbf{x})$ as features in a new feature space, which we call the *intermediate feature space*. The final decision is made by another classifier that takes the intermediate feature space as input and outputs a class label. In Ref. [149] this class of methods is named "class-indifferent." Of course we can build layer upon layer of classifiers in such a manner. The important question is how we train such architectures to make sure that the increased complexity is justified by a corresponding gain in accuracy.

5.1 HOW DO WE GET PROBABILITY OUTPUTS?

Some of the base classifiers described in Chapter 2 produce soft labels right away. Examples of such classifiers are the linear discriminant classifier, quadratic discriminant classifier, and Parzen classifier. The values of the discriminant functions can be used as the degrees of support (in $\Re$). It is more convenient though if these degrees were in the interval [0, 1], with 0 meaning "no support" and 1 meaning "full support." We can simply normalize the values so that $\Re$ is mapped to [0, 1] and the sum of the supports is one. The standard solution to this problem is the *softmax* method [15]. Let $g_1(\mathbf{x}), \ldots, g_c(\mathbf{x})$ be the output of classifier D. Then the new support scores $g'_1(\mathbf{x}), \ldots, g'_c(\mathbf{x})$, $g'_j(\mathbf{x}) \in [0, 1]$, $\sum_{j=1}^{c} g'_j(\mathbf{x}) = 1$, are obtained as

$$g'_j(\mathbf{x}) = \frac{\exp\{g_j(\mathbf{x})\}}{\sum_{k=1}^{c} \exp\{g_k(\mathbf{x})\}} \tag{5.2}$$

It would be better if $g'_j(\mathbf{x})$ were credible estimates of the probabilities for the classes given the input $\mathbf{x}$. This section describes some ways to obtain continuous outputs as estimates of the posterior probabilities $P(\omega_j|\mathbf{x}), j = 1, \ldots, c$.

5.1.1 Probabilities Based on Discriminant Scores

5.1.1.1 Linear and Quadratic Discriminant Classifiers. For these two classifiers we assumed normal class-conditional densities (a seldom valid but very practical assumption). The discriminant function for class ω_j, $g_j(\mathbf{x})$, was derived from $\log P(\omega_j)p(\mathbf{x}|\omega_j)$ by dropping all the additive terms that did not depend on the class label. Let C be a constant (possibly depending on $\mathbf{x}$ but not on j) absorbing

all additive terms that were dropped and let $A = \exp\{C\}$. Then

$$P(\omega_j)p(\mathbf{x}|\omega_j) = A\exp\{g_j(\mathbf{x})\} \tag{5.3}$$

The posterior probability for class ω_j for the given $\mathbf{x}$ is

$$P(\omega_j|\mathbf{x}) = \frac{P(\omega_j)p(\mathbf{x}|\omega_j)}{p(\mathbf{x})} \tag{5.4}$$

$$= \frac{A \times \exp\{g_j(\mathbf{x})\}}{\sum_{k=1}^{c} A \times \exp\{g_k(\mathbf{x})\}} = \frac{\exp\{g_j(\mathbf{x})\}}{\sum_{k=1}^{c} \exp\{g_k(\mathbf{x})\}} \tag{5.5}$$

which is the softmax transform (5.2). For the two-class problem, there is only one discriminant function $g(\mathbf{x})$, which we compare with the threshold 0. Therefore we may take $g_1(\mathbf{x}) = g(\mathbf{x})$ and $g_2(\mathbf{x}) \equiv 0$. Applying Eq. (5.2), we obtain

$$g_1'(\mathbf{x}) = \frac{\exp\{g_1(\mathbf{x})\}}{\exp\{g_1(\mathbf{x})\} + 1} = \frac{1}{1 + \exp\{-g_1(\mathbf{x})\}} \tag{5.6}$$

and $g_2'(\mathbf{x}) = 1 - g_1'(\mathbf{x})$. This is also called the *logistic link function* [150].

Alternatively, we can think of $g(\mathbf{x})$ as a new feature and estimate in this new one-dimensional space the two class-conditional pdfs, $p(g(\mathbf{x})|\omega_1)$ and $p(g(\mathbf{x})|\omega_2)$. The posterior probabilities are calculated from the Bayes formula. The same approach can be extended to a c-class problem by estimating a class-conditional pdf $p(g_j(\mathbf{x})|\omega_j)$ on $g_j(\mathbf{x})$, $j = 1, \ldots, c$, and calculating subsequently $P(\omega_j|\mathbf{x})$ using all $p(g_k(\mathbf{x})|\omega_k)$ and $P(\omega_k)$, $k = 1, \ldots, c$.

5.1.1.2 Neural Networks and Kernel Classifiers. Consider a neural network (NN) with c outputs, each corresponding to a class. Denote the output by $(y_1, \ldots, y_c) \in \Re^c$ and the target by $(t_1, \ldots, t_c) \in \{0,1\}^c$. The target for an object $\mathbf{z}_j$ from the data set $\mathbf{Z}$ is typically a binary vector with 1 at the position of the class label of $\mathbf{z}_j$ and zeros elsewhere. It is known that if trained to optimize the squared error between the NN output and the target, in an ideal asymptotical case, the NN output y_j will be an approximation of the posterior probability $P(\omega_j|\mathbf{x})$, $j = 1, \ldots, c$ [27,29]. Wei et al. [151] argue that the theories about the approximation are based on several assumptions that might be violated in practice: (1) that the network is sufficiently complex to model the posterior distribution accurately, (2) that there are sufficient training data, and (3) that the optimization routine is capable of finding the global minimum of the error function. The typical transformation that forms a probability distribution from $(y_1, \ldots, y_L)$ is the softmax transformation (5.2) [15]. To make this distribution more realistic Wei et al. [151] suggest a histogram-based remapping function. The parameters of this function are tuned separately from the NN training. In the operation phase, the NN output is fed to the remapping function and calibrated to give more adequate posterior probabilities.

A similar method called the *bin method* has been proposed for calibrating the outputs from the naive Bayes classifier [152] and has also been applied for support vector machine classifiers (SVM) [153]. The method considers one discriminant score at a time, for example, $g_j(\mathbf{x})$. The discriminant scores for all the training examples are sorted and placed into b bins of equal size. Using the true labels of the training examples we estimate the posterior probability for ω_j within each bin. Thus the classifier output is discretized. The assigned posterior probabilities are recovered from the respective bin for the concrete value of $g_j(\mathbf{x})$. This discretized output is less detailed but hopefully more accurate than a simply rescaled $g_j(\mathbf{x})$ using softmax.

For the kernel group of classifiers, an example of which is the Parzen classifier, we often have the ready-made probability estimates coming from the kernel estimates of the class-conditional pdfs, for example, Eq. (2.35).

5.1.2 Probabilities Based on Counts: Laplace Estimator

Consider finding probability estimates from decision trees. Each leaf of the tree defines a set of posterior probabilities. These are assigned to any point that reaches the leaf. The difficulties in obtaining good estimates of the posterior probabilities have been addressed in several publications [154]. Provost and Domingos [154] analyze the reasons for the insufficient capability of the standard decision tree classifiers to provide adequate estimates of the probabilities and conclude that the very heuristics that help us build small and accurate trees are responsible for that. Special amendments were proposed that led to the so-called *probability estimating trees* (PETs). These trees still have high classification accuracy, but their main purpose is to give more accurate estimates of the posterior probabilities.

We calculate estimates of $P(\omega_j|\mathbf{x})$, $j = 1, \ldots, c$, as the class proportions of the training data points that reached the leaf (the *maximum likelihood* (ML) estimates). Let $k_1, \ldots, k_c$ be the number of training points from classes $\omega_1, \ldots, \omega_c$, respectively, at some leaf node t, and let $K = k_1 + \cdots + k_c$. The ML estimates are

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{k_j}{K}, \qquad j = 1, \ldots, c \tag{5.7}$$

The problem is that when the total number of points, K, is small, the estimates of these probabilities are unreliable. Besides, the tree growing strategies try to make the leaves as pure as possible. Thus most probability estimates will be pushed towards 1 and 0 [152].

To remedy this, the *Laplace estimate* or *Laplace correction* can be applied [152,154,155]. The idea is to adjust the estimates so that they are less extreme. For c classes, the Laplace estimate used in Ref. [154] is

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{k_j + 1}{K + c} \tag{5.8}$$

Zadrozny and Elkan [152] apply a different version of the Laplace estimate using a parameter m that controls the degree of regularization of the estimate (called

m-estimation). The idea is to smooth the posterior probability towards the (estimate of the) prior probability for the class

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{k_j + m \times \hat{P}(\omega_j)}{K + m} \tag{5.9}$$

If m is large, then the estimate is close to the prior probability. If $m = 0$, then we have the ML estimates and no regularization. Zadrozny and Elkan suggest that m should be chosen so that $m \times P(\omega_j) \approx 10$ and also point out that practice has shown that the estimate (5.9) is quite robust with respect to the choice of m.

Suppose that ω^* is the majority class at node t. Ting and Witten [155] propose the following version of the Laplace estimator

$$\hat{P}(\omega_j|\mathbf{x}) = \begin{cases} 1 - \dfrac{\sum_{l \neq j} k_l + 1}{K + 2}, & \text{if } \omega_j = \omega^*, \\ [1 - P(\omega^*|\mathbf{x})] \times \dfrac{k_j}{\sum_{l \neq j} k_l}, & \text{otherwise.} \end{cases} \tag{5.10}$$

The general consensus in the PET studies is that for good estimates of the posterior probabilities, the tree should be grown without pruning and a form of Laplace correction should be used for calculating the probabilities.

The same argument can be applied for smoothing the estimates of the k nearest neighbor classifier (k-nn) and the histogram classifier both discussed in Chapter 2. There are many weighted versions of k-nn whereby the posterior probabilities are calculated using distances. While the distance-weighted versions have been found to be asymptotically equivalent to the nonweighted versions in terms of classification accuracy [72], there is no such argument when class ranking is considered. It is possible that the estimates of the "soft" k-nn versions are more useful for ranking than for labeling. A simple way to derive $\hat{P}(\omega_j|\mathbf{x})$ from k-nn is to average the similarities between $\mathbf{x}$ and its nearest neighbors from class ω_j. Let k be the number of neighbors, $\mathbf{x}^{(i)}$ be the ith nearest neighbor of $\mathbf{x}$, and $d(\mathbf{x}, \mathbf{x}^{(i)})$ be the distance between $\mathbf{x}$ and $\mathbf{x}^{(i)}$. Then

$$\hat{P}(\omega_j|\mathbf{x}) = \frac{\sum_{\mathbf{x}^{(j)} \in \omega_j} \dfrac{1}{d(\mathbf{x}, \mathbf{x}^{(j)})}}{\sum_{i=1}^{k} \dfrac{1}{d(\mathbf{x}, \mathbf{x}^{(i)})}}. \tag{5.11}$$

Albeit intuitive, these estimates are not guaranteed to be good approximations of the posterior probabilities.

Example: Laplace Corrections and Distance-Based k-nn Probability Estimates. Figure 5.1 shows a point in a two-dimensional feature space (the cross, $\times$) and its seven nearest neighbors from ω_1 (open circles), ω_2 (bullets), and ω_3 (triangle).

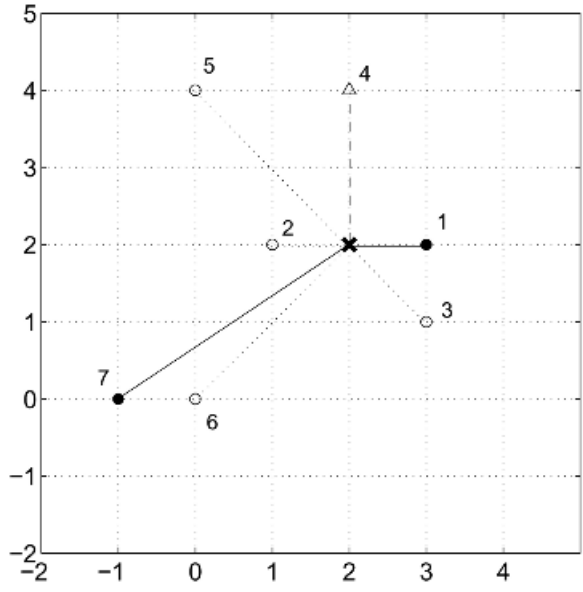


Fig. 5.1 *A point in a two-dimensional feature space and its seven nearest neighbors from* ω_1 *(open circles),* ω_2 *(bullets), and* ω_3 *(triangle).*

The Euclidean distances between $\mathbf{x}$ and its neighbors are as follows:

$\mathbf{x}$	1	2	3	4	5	6	7
Distance	1	1	$\sqrt{2}$	2	$2\sqrt{2}$	$2\sqrt{2}$	$\sqrt{13}$
Label	ω_2	ω_1	ω_1	ω_3	ω_1	ω_1	ω_2

The probability estimates $\mu_j(\mathbf{x}), j = 1, 2, 3$, using the Laplace corrections and the distance-based formula are as follows:

Method	$\mu_1(\mathbf{x})$	$\mu_2(\mathbf{x})$	$\mu_3(\mathbf{x})$
ML	$\frac{4}{7}$	$\frac{2}{7}$	$\frac{1}{7}$
Standard Laplace [154]	$\frac{5}{10}$	$\frac{3}{10}$	$\frac{2}{10}$
m-estimation [152] ($m = 12$, equiprobable classes)	$\frac{8}{19}$	$\frac{6}{19}$	$\frac{5}{19}$
Ting and Witten [155]	$\frac{12}{27}$	$\frac{10}{27}$	$\frac{5}{27}$
Distance-based	0.58	0.30	0.12

As seen in the table, all the corrective modifications of the estimates bring them closer to one another compared to the standard ML estimates, that is, the modifications smooth the estimates away from the $0/1$ bounds.

Accurate estimates are a sufficient but not a necessary condition for a high classification accuracy. The final class label will be correctly assigned as long as the degree of support for the correct class label exceeds the degrees for the other classes. Investing effort into refining the probability estimates will be justified in problems with a large number of classes c where the ranking of the classes by their likelihood is more important than identifying just one winning label. There is an increasing stream of such applications, for example, person identification, text categorization, fraud detection, and so on.

5.2 CLASS-CONSCIOUS COMBINERS

5.2.1 Nontrainable Combiners

"Nontrainable" means that the combiner has no extra parameters that need to be trained; that is, the ensemble is ready for operation as soon as the base classifiers are trained. Simple nontrainable combiners calculate the support for class ω_j using only the jth column of $DP(\mathbf{x})$ by

$$\mu_j(\mathbf{x}) = \mathcal{F}[d_{1,j}(\mathbf{x}), \ldots d_{L,j}(\mathbf{x})] \tag{5.12}$$

where $\mathcal{F}$ is a combination function. The class label of $\mathbf{x}$ is found as the index of the maximum $\mu_j(\mathbf{x})$. The combination function $\mathcal{F}$ can be chosen in many different ways. The most popular choices are:

- *Simple mean* (*average*) ($\mathcal{F} =$ average).

$$\mu_j(\mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L} d_{i,j}(\mathbf{x}) \tag{5.13}$$

- *Minimum/maximum/median* ($\mathcal{F} =$ minimum/maximum/median). For example,

$$\mu_j(\mathbf{x}) = \max_i \{d_{i,j}(\mathbf{x})\} \tag{5.14}$$

- *Trimmed mean* (*competition jury*). For a K percent trimmed mean the L degrees of support are sorted and K percent of the values are dropped on each side. The overall support $\mu_j(\mathbf{x})$ is found as the mean of the remaining degrees of support.
- *Product* ($\mathcal{F} =$ product).

$$\mu_j(\mathbf{x}) = \prod_{i=1}^{L} d_{i,j}(\mathbf{x}) \tag{5.15}$$

- *Generalized mean* [156]

$$\mu_j(\mathbf{x}, \alpha) = \left(\frac{1}{L} \sum_{i=1}^{L} d_{i,j}(\mathbf{x})^{\alpha} \right)^{1/\alpha} \tag{5.16}$$

with the following special cases:

$$\alpha \to -\infty \Rightarrow \mu_j(\mathbf{x}, \alpha) = \min_i \{d_{i,j}(\mathbf{x})\} \quad \text{(minimum)}$$

$$\alpha = -1 \Rightarrow \mu_j(\mathbf{x}, \alpha) = \left(\frac{1}{L} \sum_{i=1}^{L} \frac{1}{d_{i,j}(\mathbf{x})} \right)^{-1} \quad \text{(harmonic mean)}$$

$$\alpha = 0 \Rightarrow \mu_j(\mathbf{x}, \alpha) = \left(\prod_{i=1}^{L} d_{i,j}(\mathbf{x}) \right)^{1/L} \quad \text{(geometric mean)}$$

$$\alpha = 1 \Rightarrow \mu_j(\mathbf{x}, \alpha) = \frac{1}{L} \sum_{i=1}^{L} d_{i,j}(\mathbf{x}) \quad \text{(arithmetic mean)}$$

$$\alpha \to \infty \Rightarrow \mu_j(\mathbf{x}, \alpha) = \max_i \{d_{i,j}(\mathbf{x})\} \quad \text{(maximum)}$$

The operation of simple combiners is illustrated diagrammatically in Figure 5.2.

Note that the geometric mean is equivalent to the product combiner as raising to the power of $1/L$ is a monotone transformation that does not depend on the class label j and therefore will not change the order of $\mu_j(\mathbf{x})$; the "winning" label obtained from the product combiner will be the same as the winning label from the geometric mean combiner.

Example: Simple Nontrainable Combiners. The following example helps to clarify simple combiners. Let $c = 3$ and $L = 5$. Assume that for a certain $\mathbf{x}$

$$DP(\mathbf{x}) = \begin{bmatrix} 0.1 & 0.5 & 0.4 \\ 0.0 & 0.0 & 1.0 \\ 0.4 & 0.3 & 0.4 \\ 0.2 & 0.7 & 0.1 \\ 0.1 & 0.8 & 0.2 \end{bmatrix} \tag{5.17}$$

Applying the simple combiners columnwise, we obtain as $(\mu_1(\mathbf{x}), \mu_2(\mathbf{x}), \mu_3(\mathbf{x}))$

$$\begin{array}{ll} \text{Average} & = (0.16, 0.46, 0.42); \\ \text{Minimum} & = (0.0, 0.0, 0.1); \\ \text{Maximum} & = (0.4, 0.8, 1.0); \\ \text{Median} & = (0.1, 0.5, 0.4); \\ \text{20\% trimmed mean} & = (0.13, 0.50, 0.33); \\ \text{Product} & = (0.0, 0.0, 0.0032). \end{array} \tag{5.18}$$

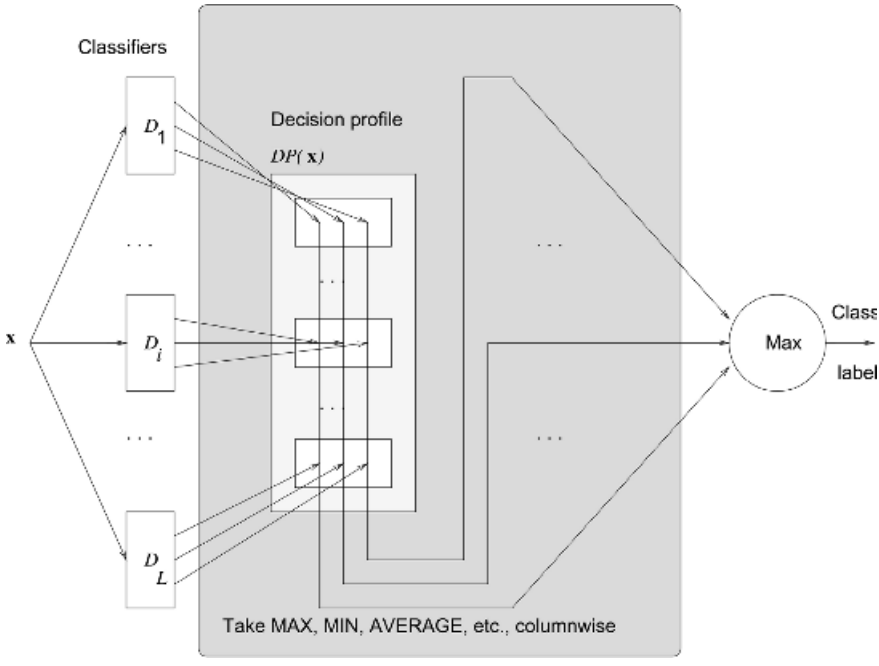


Fig. 5.2 *Operation of simple nontrainable (class-conscious) combiners.*

Note that we do not require that $d_{i,j}(\mathbf{x})$ for classifier D_i sum to one. We only assume that they are measures in the same units. If we take the class with the maximum support to be the class label of $\mathbf{x}$, the minimum, maximum, and product will label $\mathbf{x}$ in class ω_3, whereas the average, the median, and the trimmed mean will put $\mathbf{x}$ in class ω_2.

The generalized mean does have an extra parameter α. As we are considering nontrainable combiners here, we assume that the system designer chooses α beforehand. This parameter can be thought of as the "level of optimism" of the combiner. The *minimum* combiner ($\alpha \to -\infty$) is the most pessimistic choice, that is, we know that ω_j is supported by *all* members of the ensemble at least as much as $\mu_j(\mathbf{x})$. At the other extreme, if we choose the *maximum* combiner, then we are most optimistic, that is, we are happy to accept an ensemble degree of support $\mu_j(\mathbf{x})$ on the ground that *at least one* member of the team supports ω_j with this degree. Therefore α controls the *level of optimism*. If we choose to tune α with respect to the ensemble performance, then we should regard the generalized mean combiner as a trainable combiner as discussed in the next subsection.

Example: Illustration of the Effect of the Level of Optimism α. To illustrate the effect of the level of optimism α we used the two-dimensional rotated check-board

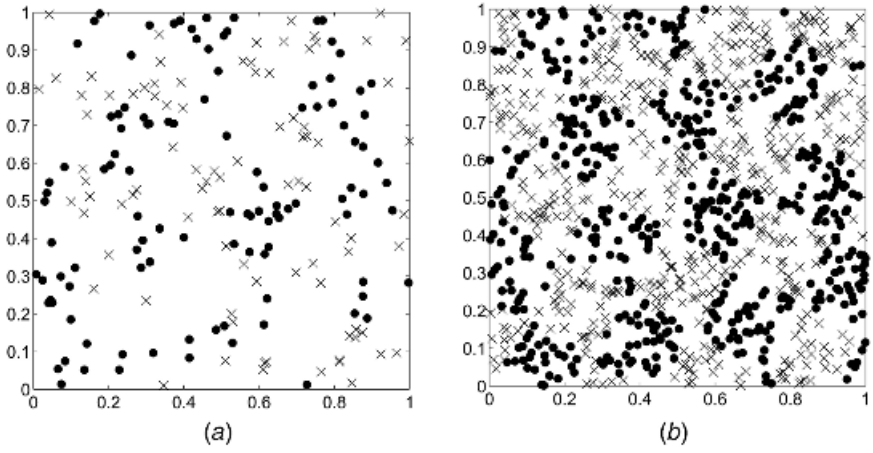


Fig. 5.3 *An example of a training* (*a*) *and testing* (*b*) *set for the rotated check-board data. One hundred randomly sampled training/testing sets were used in the experiment.*

data set. The training and testing data sets are shown in Figure 5.3*a* and *b*, respectively.

One hundred training/testing sets were generated from a uniform distribution within the unit square. The labels to the points were assigned as in the rotated check-board example in Section 1.5.4. In each experiment, the training set consisted of 200 examples and the testing set consisted of 1000 examples. Each ensemble was formed by taking 10 bootstrap samples of size 200 from the training data (uniform sampling with replacement) and training a classifier on each sample. The Parzen classifier was used from the PRTOOLS toolbox for Matlab.[23] The generalized mean formula (5.16) was used where the level of optimism α was varied from -50 to 50 with finer discretization from -1 to 1. The ensemble error averaged across the 100 runs is plotted against α in Figure 5.4.

A zoom window of the ensemble error for $\alpha \in [-2, 5]$ is shown in Figure 5.5. The simple mean, product, and harmonic mean combiners are identified on the curve. For this example the simple mean combiner gave the best result.

The results from the illustration above should not be taken as evidence that the mean combiner is always the best. The shape of the curve will depend heavily on the problem and on the base classifier used. The *average* and the *product* are the two most intensively studied combiners. Yet, there is no guideline as to which one is better for a specific problem. The current understanding is that the average, in general, might be less accurate than the product for some problems but is the more stable of the two [11,157–160].

[23] The smoothing parameter of the Gaussian kernel does not have to be specified in the Parzen classifier version in PRTOOLS. This parameter is tuned within the classification routine using the training data.

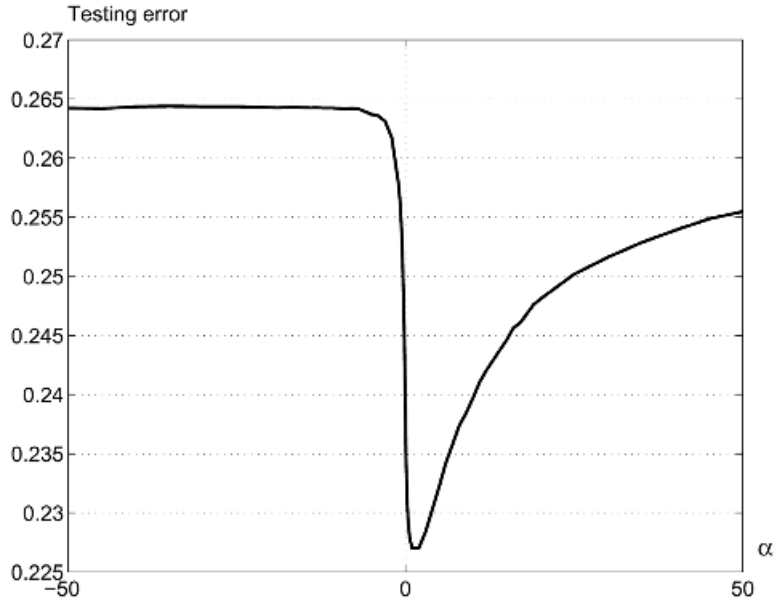


Fig. 5.4 *Ensemble error for the generalized mean combiner for* $\alpha \in [-50, 50]$.

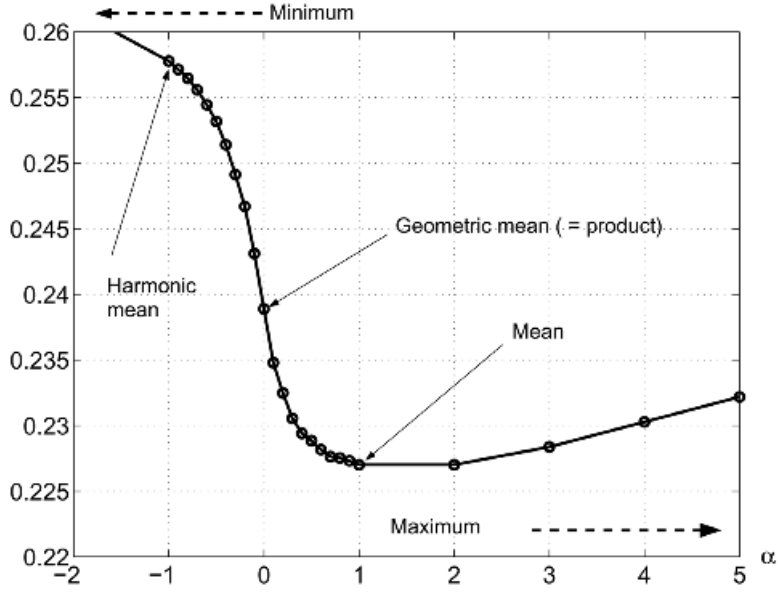


Fig. 5.5 *A zoom on the ensemble error for the generalized mean combiner for* $\alpha \in [-2, 5]$.

The *ordered weighted averaging* (OWA) is another generalized model from the nontrainable group [161]. We use a set of L coefficients, one for each classifier in the ensemble. Ordered weighted averaging does not attach a specific coefficient to a classifier. First the classifier outputs for ω_j (jth column of $DP(\mathbf{x})$) are arranged in a descending order and then a weighted sum is calculated using the coefficient associated with the *place* in this ordering.

Let $\mathbf{b} = [b_1, \ldots, b_L]^T$ be a vector with coefficients such that

$$\sum_{k=1}^{L} b_k = 1, \qquad 0 \le b_k \le 1, \quad k = 1, \ldots, L$$

The support for ω_j is calculated as the dot product of $\mathbf{b}$ and the vector

$$[d_{i_1,j}(\mathbf{x}), \ldots, d_{i_L,j}(\mathbf{x})]^T$$

where $i_1, \ldots, i_L$ is a permutation of the indices $1, \ldots, L$, such that

$$d_{i_1,j}(\mathbf{x}) \ge d_{i_2,j}(\mathbf{x}) \ge \cdots \ge d_{i_L,j}(\mathbf{x})$$

Thus the support for ω_j is

$$\mu_j(\mathbf{x}) = \sum_{k=1}^{L} b_k \times d_{i_k,j}(\mathbf{x}) \tag{5.19}$$

Example: Ordered Weighted Averaging Combiners. When a jury has to assess a sport performance (e.g., in gymnastics, acrobatics, ice-skating), to avoid, or at least reduce, subjective bias, usually the highest and the lowest marks are dropped, and the remaining $L - 2$ marks are averaged (the trimmed mean/competition jury combiner). Let the support for class ω_j be $[0.6, 0.7, 0.2, 0.6, 0.6]^T$. To implement the competition jury combiner using OWA aggregation we assign coefficients $\mathbf{b} = [0, 1/3, 1/3, 1/3, 0]^T$. This yields

$$\mu_1(u) = \left[0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0\right][0.7, 0.6, 0.6, 0.6, 0.2]^T = 0.6 \tag{5.20}$$

By selecting $\mathbf{b}$, a number of operations can be modeled, and further operations can be created

- Minimum: $\mathbf{b} = [0, 0, \ldots, 0, 1]^T$.
- Maximum: $\mathbf{b} = [1, 0, \ldots, 0, 0]^T$.
- Average: $\mathbf{b} = [1/L, 1/L, \ldots, 1/L]^T$.
- Competition jury (trimmed mean): $\mathbf{b} = [0, 1/(L-2), \ldots, 1/(L-2), 0]^T$.

The coefficient vector **b** can be either designed in advance or found from data. Yager and Filev [162] show how **b** can be designed to model linguistic quantifiers such as *almost all*, *few*, *many*, *most*, *nearly half*, and so on.

Various other aggregation connectives from fuzzy set theory can be used as the combiner [156,163,164]. The main question is *when* we should use one formula or another. It seems that we are so busy developing new aggregation connectives that a deeper analysis of the already rich toolbox is being pushed to the side.

5.2.2 Trainable Combiners

5.2.2.1 The Weighted Average. Various weighted average combiners have been proposed in the literature. Three groups can be distinguished based on the number of weights:

- *L weights*. In this model there is one weight per classifier. The support for class ω_j is calculated as

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} w_i \, d_{i,j}(\mathbf{x}) \tag{5.21}$$

  The weight for classifier D_i is usually based on its estimated error rate [165].
- $c \times L$ *weights*. The support for class ω_j is calculated as

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} w_{ij} \, d_{i,j}(\mathbf{x}) \tag{5.22}$$

  Here the weights are specific for each class. Only the jth column of the decision profile is used in the calculation, that is, the support for class ω_j is obtained from the individual supports for ω_j (class-conscious combiners). Linear regression is the commonly used procedure to derive the weights for this model [155,166–169].
- $c \times c \times L$ *weights*. The support for each class is obtained by a linear combination of all of the decision profile $DP(\mathbf{x})$

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} \sum_{k=1}^{c} w_{ikj} \, d_{i,k}(\mathbf{x}) \tag{5.23}$$

  where w_{ikj} is the (i, k)th weight for class ω_j. The whole of the decision profile is used as the intermediate feature space (class-indifferent combiners).

In this section we will consider the class-conscious combiner (5.22). We look at $d_{i,j}(\mathbf{x})$, $i = 1, \ldots, L$, as L point estimates of the same uncertain quantity $P(\omega_j|\mathbf{x})$. If the estimates are unbiased, then we can form a nonbiased, minimum variance esti-

mate $\mu_j(\mathbf{x})$ of $P(\omega_j|\mathbf{x})$ by taking the weighted average (5.22) and restricting the coefficients w_i to sum up to one [169]

$$\sum_{i=1}^{L} w_i = 1 \tag{5.24}$$

Note that we may or may not require that the coefficients are nonnegative. The weights are derived so that they minimize the variance of $\mu_j(\mathbf{x})$. Using these weights, the aggregated estimate is guaranteed to have variance at most as large as the variance of any of the classifiers in the ensemble. Since we assumed that the estimators are unbiased, the variance of each of the estimates $d_{i,j}(\mathbf{x})$, $i = 1, \ldots, L$, is equivalent to its expected squared error. Hence the weighted average (5.22) with coefficients found by minimizing the variance will be a better estimate of the posterior probability $P(\omega_j|\mathbf{x})$ than any of the ensemble members. This is a highly desirable property, but, as discussed before, poorer estimates might lead to better classification than more accurate estimates as long as the class with the highest true probability has the top rank. High accuracy of the estimates is a sufficient but not a necessary condition for high classification accuracy. Therefore the regression methods discussed here might not be the most appropriate training scenario for combining classifier outputs.

Example: Variance of the Estimate of $P(\omega_j|\mathbf{x})$. For regression problems we seek to approximate a continuous-valued output. For classification problems the target output is given only in the form of a class label. So the target values for $P(\omega_j|\mathbf{x})$ provided with the data set are either 1 (in ω_j) or 0 (not in ω_j). If we accept a model *without noise* then we trust the $0/1$ values to be the true posterior probabilities at the data points in the given data set $\mathbf{Z}$. Table 5.1 shows a hypothetical output for class ω_1 of a classifier ensemble $\mathcal{D} = \{D_1, D_2\}$ for a data set $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{10}\}$. The first three points in the data set have labels ω_1 and the remaining seven have a different class label. To form the table, the first columns of the 10 decision profiles, $DP(\mathbf{z}_1), \ldots, DP(\mathbf{z}_{10})$, are taken so that only the support for ω_1 is shown.

The *variance* of the error of D_i considered here is the variance of the *approximation* error, not the classification error. The approximation error of D_1 has the following 10 values

$$(1 - 0.71) \quad (1 - 0.76) \quad (1 - 0.15) \quad -0.09 \quad -0.15 \quad -0.62 \quad -0.98 \quad -0.56 \quad -0.44 \quad -0.79$$

TABLE 5.1 Hypothetical Output for ω_1 from Two Classifiers for a Data Set $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{10}\}$.

Data Point	$\mathbf{z}_1$	$\mathbf{z}_2$	$\mathbf{z}_3$	$\mathbf{z}_4$	$\mathbf{z}_5$	$\mathbf{z}_6$	$\mathbf{z}_7$	$\mathbf{z}_8$	$\mathbf{z}_9$	$\mathbf{z}_{10}$
$d_{1,1}(\mathbf{z}_k)$	0.71	0.76	0.15	0.09	0.15	0.62	0.98	0.56	0.44	0.79
$d_{2,1}(\mathbf{z}_k)$	0.41	0.27	0.91	0.15	0.64	0.90	0.68	0.95	0.22	0.14
Target (ω_1)	1	1	1	0	0	0	0	0	0	0

The mean of the error of D_1 is -0.225 and the variance of the error of D_1 is

$$\sigma_1^2 = \frac{1}{10-1}((1-0.71+0.225)^2 + (1-0.76+0.225)^2 + \cdots + (-0.79+0.225)^2) \approx 0.32 \tag{5.25}$$

The covariance matrix of the approximation errors of the two classifiers is

$$\Sigma = \begin{bmatrix} 0.32 & 0.22 \\ 0.22 & 0.34 \end{bmatrix} \tag{5.26}$$

Assume that we round the prediction so as to count the classification errors with respect to ω_1. From the number of misclassified elements of $\mathbf{Z}$, the classification error of D_1 is 50 percent and the classification error of D_2 is 60 percent.

One version of *constrained regression* for finding the weights that minimize the variance of Eq. (5.22) is derived by assuming that the expert's errors *in approximating the posterior probability*, $P(\omega_j|\mathbf{x}) - d_{i,j}(\mathbf{x})$, are normally distributed with zero mean [169]. Denote by σ_{ik} the covariance between the approximation errors by classifiers D_i and D_k. The weights are found by minimizing the following objective function which includes a Lagrangean term for enforcing the constraint (5.24)

$$J = \sum_{i=1}^{L}\sum_{k=1}^{L} w_i w_k \sigma_{ik} - \lambda\left(\sum_{i=1}^{L} w_i - 1\right) \tag{5.27}$$

The solution minimizing J is

$$\mathbf{w} = \Sigma^{-1}\mathbf{I}(\mathbf{I}^T\Sigma^{-1}\mathbf{I})^{-1} \tag{5.28}$$

where $\mathbf{w} = [w_1, \ldots, w_L]^T$ is the vector of weights, Σ is the covariance matrix for the classifiers' approximation errors and $\mathbf{I}$ is an L-element vector with ones.

For the above example,

$$\mathbf{w} = \begin{bmatrix} 5.6 & -3.6 \\ -3.6 & 5.3 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\left(\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 5.6 & -3.6 \\ -3.6 & 5.3 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)^{-1} = \begin{bmatrix} 0.54 \\ 0.46 \end{bmatrix} \tag{5.29}$$

We should keep in mind that all the weights and covariances have to be labeled additionally by another index, say j, to specify which of the c posterior probabilities we are approximating. To simplify notation this index is not shown here. The constrained regression method for combining the outputs of the ensemble will produce an $L \times c$ matrix with weights $\mathbf{W} = \{w_{ij}\}$, where w_{ij} will be the weight of classifier D_i for approximating probability $P(\omega_j|\mathbf{x})$. Owing to the weight constraint each column of $\mathbf{W}$ will sum to 1.

Example: Constrained Regression for Calculating Classifier's Weights. The example below shows the results from an experiment with the banana data. The

ensemble size was varied from $L = 2$ to $L = 30$. For each ensemble size, one hundred data sets have been generated with 100 data points for training (50 on each banana) and additional 1000 point for testing (500 on each banana). Linear classifiers have been trained on L bootstrap samples of the training data and combined using the weighted average whereby the weights have been calculated through the solution of the constrained regression problem (5.28). For comparison, the simple average combiner was applied as well.

Plotted in Figure 5.6 are the testing errors for the two combiners versus the ensemble size L, averaged across 100 runs. The error bars show the 95 percent confidence intervals for the error. The figure shows a remarkable difference between the two combiners. While the simple average hardly improves upon the error with increasing L, the weighted average drives the testing error to less than 1 percent. The tendency for overtraining is seen for values of L greater than 20. The large spikes for $L > 25$ indicate that the covariance matrix Σ has been close to singular and the inverse has been calculated with poor precision. Indeed, for 25 classifiers in the ensemble, there are $(25 \times 26)/2 = 325$ elements of Σ to be estimated. The training data set consists of 100 data points, so spurious results can be expected.

Bootstrap sampling is commonly known as *bagging*. However, bagging has been devised for label outputs and the combination is done by taking the majority vote. It has been found that traditional bagging does not work well with linear classifiers as the base classifiers. In this example, the bagging strategy coupled with the constrained regression combiner outperformed the simple average by a large margin

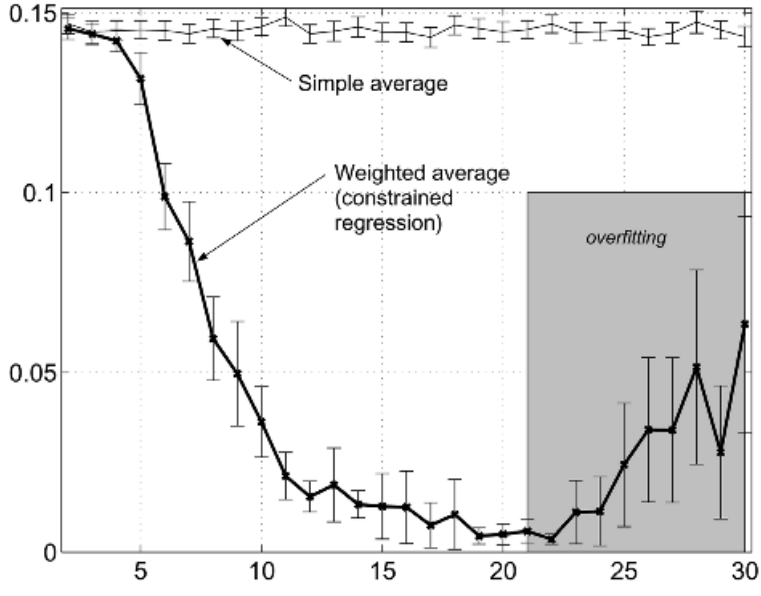


Fig. 5.6 *Weighted average (constrained regressions) and simple average on the banana data. The error bars mark the 95 percent confidence interval for the classification error calculated on the basis of 100 runs.*

before taking a turn for overfitting. Curiously, the training error pattern matches the testing error pattern shown in Figure 5.6 very closely. As soon as the problems with the inverse of Σ start occurring, the training should be stopped and the best ensemble should be selected on the basis of its training success. This illustration comes to show that there are possibilities for improvement in the "classical" methods that are worthy of further exploration.

Suppose that the classifier outputs for class ω_j were independent. Then Σ in Eq. (5.28) is diagonal with the variances of $D_1, \ldots, D_L$ along the diagonal. In this case the weights are proportional to the inverse of the variances $w_i \propto (1/\sigma_i^2)$, and Eq. (5.28) reduces to

$$w_i = \frac{\frac{1}{\sigma_i^2}}{\sum_{k=1}^{L} \frac{1}{\sigma_k^2}} \tag{5.30}$$

For classification purposes, the value of $\mu_j(\mathbf{x})$ does not have to be an unbiased estimate of the posterior probability, therefore the weights do not have to be constrained by Eq. (5.24). Also, an intercept term can be added. It has been found that there is not much difference between the performance of the combiner with or without an intercept term [155,169]. Restricting the weights to be nonnegative has not been found to either improve or degrade the performance of the combiner. Hence Ting and Witten suggest that nonnegative weights are preferable for interpretation purposes [155]. The larger the weight, the more important the classifier.

Minimum squared error is the traditional criterion for regression [166–168,170]. Minimizing the squared error leads to improved classification in an indirect way: through approximation of the posterior probabilities. It is more natural to try and minimize directly the classification error. The trouble is that there is no easy analytical solution for finding the weights. One possibility is to treat the combination of the classifier outputs as a standard pattern recognition problem and apply linear discriminant analysis to it [171]. If we use the whole decision profile as the input features, then we have a *class-indifferent* combiner. However, if we only want to use the jth column of $DP(\mathbf{x})$, containing the support for ω_j, then each discriminant function will be based on its own set of (intermediate) features. The discriminant score for class ω_j, $g_j(\mathbf{x})$, is calibrated so as to estimate $P(\omega_j | j\text{th column of } DP(\mathbf{x}))$. The maximum posterior probability will give the label of $\mathbf{x}$. Linear discriminant analysis is only one of many possible classifiers that can be built on the intermediate feature space. Any standard pattern classifier can be attempted. Ueda [171] uses a *probabilistic descent* method to derive the weights for combining neural networks as the base classifiers. Some authors consider using *genetic algorithms* for finding the weights [172,173].

5.2.2.2 Fuzzy Integral. Fuzzy integral (FI) has been reported to give excellent results as a classifier combiner [163,164,174–178]. Here we will only present the algorithm without taking a detour into fuzzy set theory.

The philosophy of the fuzzy integral combiner is to measure the "strength" not only for each classifier alone but also for all the subsets of classifiers. Every subset of classifiers has a measure of strength that expresses how good this *group* of experts is for the given input $\mathbf{x}$. The ensemble support for ω_j, $\mu_j(\mathbf{x})$, is obtained from the support values $d_{i,j}(\mathbf{x})$, $i = 1, \ldots, L$, by taking into account the competences of the groups of the various subsets of experts. The measure of strength of the subsets is called a *fuzzy measure*. The concept of fuzzy measure and the operation of fuzzy integral is explained alongside the example below.

Example: Fuzzy Measure. Let $\mathcal{D} = \{D_1, D_2, D_3\}$, and let the fuzzy measure g be defined as below:

Subset	D_1	D_2	D_3	D_1, D_2	D_1, D_3	D_2, D_3	D_1, D_2, D_3
g	0.3	0.1	0.4	0.4	0.5	0.8	1

Suppose that the jth column of the decision profile of the current $\mathbf{x}$ is $[0.1, 0.7, 0.5]^T$. These are the values of support given by the three classifiers to the hypothesis that $\mathbf{x}$ comes from ω_j. To calculate the overall support $\mu_j(\mathbf{x})$ we first sort the degrees of support in ascending order. For illustration purposes we append 0 at the beginning and 1 at the end of the list if these values are not already present there. For each different value α in the list we identify which classifiers in the ensemble have given support for ω_j that is greater than or equal to α. The subset of such classifiers is called and α-cut in fuzzy set terminology, denoted here H_α. Below we show the list of αs with the appended 0 and 1, the subsets of classifiers H_α (the α-cuts) and the fuzzy measure of each subset H_α:

$$
\begin{aligned}
&\alpha = 0, && H_0 = \{D_1, D_2, D_3\}; && g(H_0) = 1\\
&\alpha = 0.1, && H_{0.1} = \{D_1, D_2, D_3\}; && g(H_{0.1}) = 1\\
&\alpha = 0.5, && H_{0.5} = \{D_2, D_3\}; && g(H_{0.5}) = 0.8\\
&\alpha = 0.7, && H_{0.7} = \{D_2\}; && g(H_{0.7}) = 0.1\\
&\alpha = 1, && H_1 = \emptyset. && g(H_1) = 0
\end{aligned}
$$

To get $\mu_j(\mathbf{x})$ we "fuse" the values of α (support) and g (strength or competence). For example, the competence of the whole set of classifiers is 1 and all the experts agree that the support for ω_j should be 0.1 or more. However, only D_2 and D_3 "think" that the support should be higher. We have to take into account their combined competence and weigh it against the highest support value that they agree upon. The ensemble support for ω_j calculated through *Sugeno fuzzy integral* is

$$\mu_j(\mathbf{x}) = \max_{\alpha} \{\min(\alpha, g(H_\alpha))\} \tag{5.31}$$

$$
\begin{aligned}
&= \max \{\min(0, 1),\ \min(0.1, 1),\ \min(0.5, 0.8),\\
&\qquad \min(0.7, 0.1),\ \min(1, 0)\}\\
&= \max \{0, 0.1, 0.5, 0.1, 0\} = 0.5.
\end{aligned} \tag{5.32}
$$

The fundamental problem with using the fuzzy integral combiner is that we usually do not have a look-up table for the fuzzy measure g. The traditional solution to this problem is to calculate the so-called λ-fuzzy measure using the data. The calculation starts with selecting a set of L values g^i, representing the individual importance of classifier D_i, $i = 1, \ldots, L$. The value of λ needed for calculating g is obtained as the unique real root greater than -1 of the polynomial

$$\lambda + 1 = \prod_{i=1}^{L} (1 + \lambda g^i), \qquad \lambda \neq 0 \tag{5.33}$$

Appendix 5A shows a piece of Matlab code for calculating λ for given $g^1, \ldots, g^L$.

Once $g^1, \ldots, g^L$ are selected and λ is found, the operation of the fuzzy integral as a classifier combiner is shown in Figures 5.7 and 5.8. Instead of $g(H_\alpha)$, the value of the fuzzy measure g for the subset of classifiers H_α at α is denoted as $g(k)$. The index k varies from 1 to L. $k = 1$ corresponds to the largest support α found among the L degrees. For our example, $g(H_{0.7}) = g(1)$, $g(H_{0.5}) = g(2)$, and $g(H_{0.1}) = g(3)$. The values of g are not taken from a table but calculated as shown in Figure 5.7.

The support for ω_k, $\mu_k(\mathbf{x})$, can be thought of as a compromise between the *competence* (represented by the fuzzy measure g) and the *evidence* (represented by the kth column of the decision profile $DP(\mathbf{x})$). Notice that the fuzzy measure vector $[g(1), \ldots, g(L)]^T$ might be different for each class, and is also specific for the current $\mathbf{x}$. The fuzzy measure vectors for two classes will be the same only if the ordering of the classifier support for both classes is the same. The algorithm in Figure 5.7

Fuzzy integral for classifier fusion

1. For a given $\mathbf{x}$ sort the kth column of $DP(\mathbf{x})$ to obtain $[d_{i_1,k}(\mathbf{x}), d_{i_2,k}(\mathbf{x}), \ldots, d_{i_L,k}(\mathbf{x})]^T$, $d_{i_1,k}(\mathbf{x})$ being the highest degree of support, and $d_{i_L,k}(\mathbf{x})$, the lowest.
2. Arrange the fuzzy densities correspondingly, i.e., $g^{i_1}, \ldots, g^{i_L}$ and set $g(1) = g^{i_1}$.
3. For $t = 2$ to L, calculate recursively
$$g(t) = g^{i_t} + g(t-1) + \lambda g^{i_t} g(t-1).$$
4. Calculate the final degree of support for class ω_k by
$$\mu_k(\mathbf{x}) = \max_{t=1}^{L} \{\min\{d_{i_t,k}(\mathbf{x}), g(t)\}\}.$$

Fig. 5.7 *Fuzzy integral for classifier fusion.*

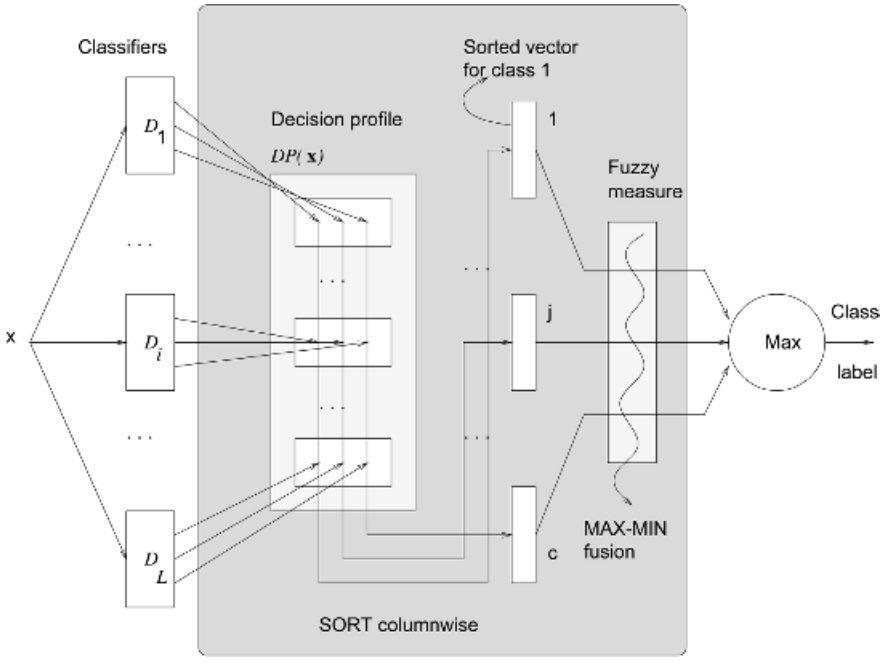


Fig. 5.8 *Operation of the fuzzy integral combiner.*

calculates the so-called *Sugeno fuzzy integral*. The second most popular type of fuzzy integral is the *Choquet fuzzy integral* [179,180]. The calculations use the same λ-fuzzy measure; the only difference is in the final equation, which should be replaced by

$$\mu_j(\mathbf{x}) = d_{i_1,k}(\mathbf{x}) + \sum_{k=2}^{L} [d_{i_{k-1},j}(\mathbf{x}) - d_{i_k,j}(\mathbf{x})] g(k-1)$$

5.3 CLASS-INDIFFERENT COMBINERS

The combiners in this group derive $\mu_j(\mathbf{x})$ using all $L \times c$ degrees of support in $DP(\mathbf{x})$. Each vector in the intermediate feature space is an expanded version of $DP(\mathbf{x})$ obtained by concatenating its L rows. Any classifier can be applied at this point, from a simple linear regression [155,171] to a neural network [181].

5.3.1 Decision Templates

The idea of the *decision templates* (DT) combiner is to remember the most typical decision profile for each class ω_j, called the *decision template*, DT_j, and then com-

pare it with the current decision profile $DP(\mathbf{x})$ using some similarity measure $\mathcal{S}$. The closest match will label $\mathbf{x}$. Figures 5.9 and 5.10 describe the training and the operation of the decision templates' combiner.

Two measures of similarity are based upon:

- *The squared Euclidean distance* ($DT(E)$). The ensemble support for ω_j is

$$\mu_j(\mathbf{x}) = 1 - \frac{1}{L \times c} \sum_{i=1}^{L} \sum_{k=1}^{c} [DT_j(i, k) - d_{i,k}(\mathbf{x})]^2 \tag{5.35}$$

where $DT_j(i, k)$ is the (i, k)th entry in decision template DT_j. The outputs μ_j are scaled to span the interval [0, 1], but this scaling is not necessary for classification purposes. We can drop the scaling coefficient $1/(L \times c)$ and the constant 1. The class with the maximal support would be the same.

If we regard $DP(\mathbf{x})$ and DT_j as vectors in the $L \times c$-dimensional intermediate feature space, the degree of support is the negative squared Euclidean distance between the two vectors. This calculation is equivalent to applying the nearest mean classifier in the intermediate feature space. While we use only the Euclidean distance in Eq. (5.35), there is no reason to stop at this choice. Any distance could be used, for example, the Minkowski, Mahalanobis, and so on.

1. **Decision templates (training)** For $j = 1, \ldots, c$, calculate the mean of the decision profiles $DP(\mathbf{z}_k)$ of all members of ω_j from the data set $\mathbf{Z}$. Call the mean *a decision template* DT_j

$$DT_j = \frac{1}{N_j} \sum_{\substack{\mathbf{z}_k \in \omega_j \\ \mathbf{z}_k \in \mathbf{Z}}} DP(\mathbf{z}_k),$$

where N_j in the number of elements of $\mathbf{Z}$ from ω_j.

2. **Decision templates (operation)** Given the input $\mathbf{x} \in \Re^n$, construct $DP(\mathbf{x})$. Calculate the **similarity** $\mathcal{S}$ between $DP(\mathbf{x})$ and each DT_j,

$$\mu_j(\mathbf{x}) = \mathcal{S}(DP(\mathbf{x}), DT_j) \quad j = 1, \ldots, c.$$

Fig. 5.9 *Training and operation of the decision templates combiner.*

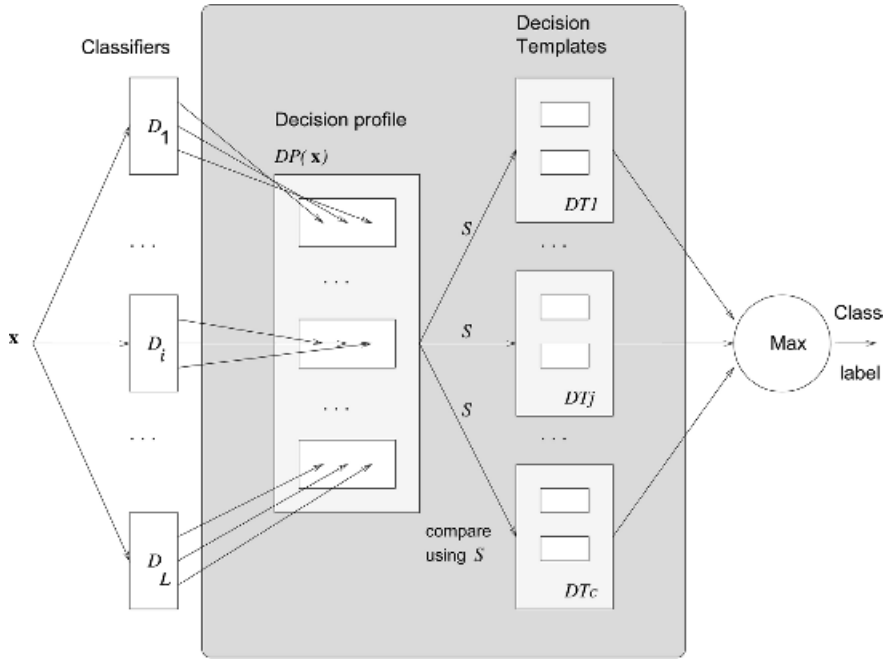


Fig. 5.10 *Operation of the decision templates combiner.*

- *A symmetric difference* ($DT(S)$). Symmetric difference comes from fuzzy set theory [163,182]. The support for ω_j is

$$\mu_j(\mathbf{x}) = 1 - \frac{1}{L \times c} \sum_{i=1}^{L} \sum_{k=1}^{c} \max \{\min \{DT_j(i,k), (1 - d_{i,k}(\mathbf{x}))\},$$
$$\min \{(1 - DT_j(i,k)), d_{i,k}(\mathbf{x})\}\} \tag{5.36}$$

Example: Illustration of the Decision Templates (*DT*) Combiner. Let $c = 3$, $L = 2$, and the decision templates for ω_1 and ω_2 be respectively

$$DT_1 = \begin{bmatrix} 0.6 & 0.4 \\ 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \quad \text{and} \quad DT_2 = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \\ 0.1 & 0.9 \end{bmatrix}$$

Assume that for an input $\mathbf{x}$, the following decision profile has been obtained:

$$DP(\mathbf{x}) = \begin{bmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}$$

The similarities and the class labels using DT(E) and DT(S) are:

DT version	$\mu_1(\mathbf{x})$	$\mu_2(\mathbf{x})$	Label
DT(E)	0.9567	0.9333	ω_1
DT(S)	0.5000	0.5333	ω_2

The difference in the opinions of the two DT versions with respect to the class label is an indication of the flexibility of the combiner.

Below we try to give more insight into why DTs are expected to work differently from other widely used combiners.

Decision templates are a class-indifferent approach because they treat the classifier outputs as a context-free set of features. All class-conscious combiners are idempotent by design, that is, if the ensemble consists of L copies of a classifier D, the ensemble decision will be no different from the decision of D. Decision templates, however, will not be necessarily identical to D; they might be better or worse.

To illustrate this point, we consider a classifier D for a two-class data set. Denote the outputs for the two classes as $d_1 = P(\omega_1|\mathbf{x}, D)$ and $d_2 = 1 - d_1 = P(\omega_2|\mathbf{x}, D)$. Taking L copies of D to form an ensemble, the decision profile for $\mathbf{x}$ contains L rows of $[d_1, d_2]$. It is not difficult to verify that all class-conscious combination methods will copy the decision of D as their final decision. However, this is not the case with DTs. Assume that we have obtained the following DTs:

$$DT_1 = \begin{bmatrix} 0.55 & \cdots & 0.45 \\ & \cdots & \\ 0.55 & & 0.45 \end{bmatrix} \quad \text{and} \quad DT_2 = \begin{bmatrix} 0.2 & \cdots & 0.8 \\ & \cdots & \\ 0.2 & & 0.8 \end{bmatrix}$$

and the decision of D for $\mathbf{x}$ is $d_1 = 0.4$ and $d_2 = 0.6$. All class-conscious methods except DTs will assign $\mathbf{x}$ to class ω_2. Using, say DT(E), we have the two Euclidean distances

$$E_1 = \sqrt{L \times [(0.55 - 0.40)^2 + (0.45 - 0.60)^2]} = \sqrt{0.045\, L} \tag{5.37}$$

$$E_2 = \sqrt{L \times [(0.2 - 0.40)^2 + (0.8 - 0.60)^2]} = \sqrt{0.080\, L} \tag{5.38}$$

Since $E_1 < E_2$, $\mathbf{x}$ will be classed in ω_1. Is this good or bad? The fact that a different classification is possible shows that DTs are not an idempotent combiner. Hence it is possible that the true label of $\mathbf{x}$ was ω_1, in which case DTs are correct where all other combiners, including D itself, are wrong. The question is in what experimental scenario should we expect DTs to be more accurate than other methods?

We ran a linear classifier on the Cleveland data set from the UCI Machine Learning Repository, using a randomly selected half of the data for training and the remaining half for testing. Every point $\mathbf{x} \in \Re^n$ can be characterized by its output values (d_1, d_2), which can be plotted as a point on the diagonal line joining points (0, 1) and (1, 0) of the unit square with axes μ_1 and μ_2.

In this illustration we abandon D and consider a distorted copy of it, denoted $\tilde{D}$, whereby the output for ω_1, ω_2 is $((d_1)^3, d_2) = ((d_1)^3, 1 - d_1)$. Let us construct an ensemble by taking L copies of $\tilde{D}$. Since the support for the two classes does not sum to one, the soft labels of the data assigned by $\tilde{D}$ are off the diagonal line, forming an arc as shown in Figure 5.11. The data points whose labels fall in the shaded area will be labeled by $\tilde{D}$ in ω_2 because for these points $d_2 > (d_1)^3$. An ensemble consisting of L copies of $\tilde{D}$ will also label the points according to the bisecting line for any class-conscious combiner.

Next we apply DT(E). The two decision templates will consist of L identical rows, therefore they also can be depicted by points in the unit square (the two crosses in Figure 5.11). Since the support for the classes is now calculated by the distance to

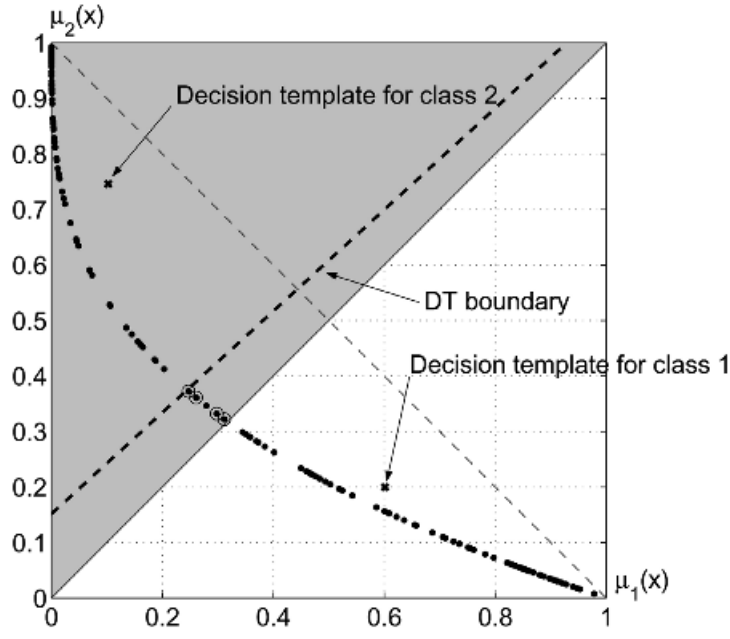


Fig. 5.11 *Illustration of the decision templates operating on an ensemble of identical distorted linear classifiers $\tilde{D}$ using the Cleveland data set. A typical run is presented. The shaded area is the decision region for classes ω_2 using $\tilde{D}$ and any class-conscious combination method. The points in the training set are depicted using their labels from $\tilde{D}$ (the arc). The two decision templates are shown with crosses and the respective new classification boundary is shown by the thick dashed line. Previously mislabeled points that are correctly labeled by the DT(E) are encircled.*

the template, a new decision boundary is found, shown by the thick dashed line. In the example shown, four points from the original training set, previously mislabeled as ω_2 are now correctly labeled as ω_1 (encircled in the figure).

This experiment was run 10,000 times with different random splits into training and testing. The training accuracy of $\tilde{D}$ was found to be 85.05 percent (with a 95 percent confidence interval (CI) [85.00 percent, 85.09 percent] calculated using Eq. (1.10)), the same as the training accuracy of any ensemble of L identical classifiers combined by any class-conscious combiner. The training accuracy of DT(E) was 86.30 percent
(95 percent CI [86.30 percent, 86.34 percent]), which is a statistically significant improvement over $\tilde{D}$ and the class-conscious combiners. The testing accuracies were, respectively:

$\tilde{D}$ and class-conscious combiners:	80.99 percent (CI = [80.94, 81.04])
DT(E):	82.96 percent (CI = [82.92, 83.01])

A statistically significant difference is observed in favor of DT(E).

This illustration was given to demonstrate that DTs are a richer combiner than the class-conscious combiners and can work even for ensembles consisting of identical classifiers. The appeal of the decision template combiner is in its simplicity. DT(E) is the equivalent to the nearest mean classifier in the intermediate feature space. Thus the DT combiner is the simplest choice from the class of class-indifferent combiners. Several studies have looked into possible applications of DTs [115,183–185].

5.3.2 Dempster–Shafer Combination

Two combination methods, which take their inspiration from the evidence combination of Dempster–Shafer (DS) theory, are proposed in Refs. [186,187]. The method proposed in Ref. [187] is commonly known as the Dempster–Shafer combiner. Here we will explain the algorithm and omit the details of how it originated from D–S theory.

As with the decision templates method, the c decision templates, $DT_1, \ldots, DT_c$ are found from the data. Instead of calculating the similarity between the decision template DT_i and the decision profile $DP(\mathbf{x})$, the following steps are carried out:

1. Let DT_j^i denote the ith row of decision template DT_j. Denote by $D_i(\mathbf{x})$ the (soft label) output of D_i, that is, $D_i(\mathbf{x}) = [d_{i,1}(\mathbf{x}), \ldots, d_{i,c}(\mathbf{x})]^T$: the ith row of the decision profile $DP(\mathbf{x})$. We calculate the "proximity" Φ between DT_j^i and the output of classifier D_i for the input $\mathbf{x}$.

$$\Phi_{j,i}(\mathbf{x}) = \frac{(1 + ||DT_j^i - D_i(\mathbf{x})||^2)^{-1}}{\sum_{k=1}^{c} (1 + ||DT_k^i - D_i(\mathbf{x})||^2)^{-1}} \tag{5.39}$$

where $\|\cdot\|$ is any matrix norm. For example, we can use the Euclidean distance between the two vectors. Thus for each decision template we have L proximities.

2. Using Eq. (5.39), we calculate for every class, $j = 1, \ldots, c$; and for every classifier, $i = 1, \ldots, L$, the following *belief* degrees

$$b_j(D_i(\mathbf{x})) = \frac{\Phi_{j,i}(\mathbf{x}) \prod_{k \neq j} (1 - \Phi_{k,i}(\mathbf{x}))}{1 - \Phi_{j,i}(\mathbf{x})[1 - \prod_{k \neq j} (1 - \Phi_{k,i}(\mathbf{x}))]} \tag{5.40}$$

3. The final degrees of support are

$$\mu_j(\mathbf{x}) = K \prod_{i=1}^{L} b_j(D_i(\mathbf{x})), \qquad j = 1, \ldots, c \tag{5.41}$$

where K is a normalizing constant.

Example: Illustration of the Dempster–Shafer Method. We use the example in Section 5.3.1. The two decision templates and the decision profile are

$$DT_1 = \begin{bmatrix} 0.6 & 0.4 \\ 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \quad DT_2 = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \\ 0.1 & 0.9 \end{bmatrix} \quad DP(\mathbf{x}) = \begin{bmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}$$

Using the Euclidean distance in Eq. (5.39), we calculate the three proximities for each decision template:

Class	$\Phi_{j,1}(\mathbf{x})$	$\Phi_{j,2}(\mathbf{x})$	$\Phi_{j,3}(\mathbf{x})$
ω_1	0.4587	0.5000	0.5690
ω_2	0.5413	0.5000	0.4310

For $c = 2$ classes, taking into account the normalization so that $\Phi_{1,i}(\mathbf{x}) + \Phi_{2,i}(\mathbf{x}) = 1$, the belief formula (5.40) for ω_1 reduces to

$$b_1(D_i(\mathbf{x})) = \frac{\Phi_{1,i}(\mathbf{x})(1 - \Phi_{2,i}(\mathbf{x}))}{1 - \Phi_{1,i}(\mathbf{x})[1 - (1 - \Phi_{2,i}(\mathbf{x}))]} \tag{5.42}$$

$$= \frac{\Phi_{1,i}(\mathbf{x})^2}{1 - \Phi_{1,i}(\mathbf{x})(1 - \Phi_{1,i}(\mathbf{x}))} \tag{5.43}$$

The value of $b_2(D_i(\mathbf{x}))$ is obtained correspondingly. The final degree of belief for class ω_j is obtained as the product of the values of b_j, $j = 1, 2$. For our example,

Class	$b_j(D_1(\mathbf{x}))$	$b_j(D_2(\mathbf{x}))$	$b_j(D_3(\mathbf{x}))$	$\mu_j(\mathbf{x})$
ω_1	0.2799	0.3333	0.4289	**0.5558**
ω_2	0.3898	0.3333	0.2462	**0.4442**

The DS combiner gives a slight preference to class ω_1.

5.4 WHERE DO THE SIMPLE COMBINERS COME FROM?

We can regard $d_{i,j}(\mathbf{x})$ as an estimate of the posterior probability $P(\omega_j|\mathbf{x})$ produced by classifier D_i. Finding an optimal (in Bayesian sense) combination of these estimates is not straightforward. Here we give a brief account of some of the theories underpinning the most common simple combiners.

5.4.1 Conditionally Independent Representations

An interesting derivation of the sum combiner (the same as the mean or average combiner) is offered in Ref. [11]. We consider L different conditionally independent feature subsets. Each subset generates a part of the feature vector, $\mathbf{x}^{(i)}$, so that $\mathbf{x} = [\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}]^T$, $\mathbf{x} \in \Re^n$. For example, suppose that there are $n = 10$ features altogether and these are grouped in the following $L = 3$ conditionally independent subsets (1,4,8), (2,3,5,6,10), and (7,9). Then

$$\mathbf{x}^{(1)} = [x_1, x_4, x_8]^T, \qquad \mathbf{x}^{(2)} = [x_2, x_3, x_4, x_6, x_{10}]^T, \qquad \mathbf{x}^{(3)} = [x_7, x_9]^T$$

From the assumed independence, the class-conditional pdf for class ω_j is a product of the class-conditional pdfs on each feature subset (representation)

$$p(\mathbf{x}|\omega_j) = \prod_{i=1}^{L} p(\mathbf{x}^{(i)}|\omega_j) \tag{5.44}$$

Deriving the product rule weighted by the prior probabilities as the optimal combiner for this case is straightforward [188–190]. The jth output of classifier D_i is an estimate of the probability

$$P(\omega_j|\mathbf{x}^{(i)}, D_i) = \frac{P(\omega_j)p(\mathbf{x}^{(i)}|\omega_j)}{p(\mathbf{x}^{(i)})} \tag{5.45}$$

hence

$$p(\mathbf{x}^{(i)}|\omega_j) = \frac{P(\omega_j|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{P(\omega_j)} \tag{5.46}$$

The posterior probability using the whole of $\mathbf{x}$ is

$$P(\omega_j|\mathbf{x}) = \frac{P(\omega_j)p(\mathbf{x}|\omega_j)}{p(\mathbf{x})} \tag{5.47}$$

$$= \frac{P(\omega_j)}{p(\mathbf{x})} \prod_{i=1}^{L} p(\mathbf{x}^{(i)}|\omega_j) \tag{5.48}$$

Substituting Eq. (5.46) into Eq. (5.48),

$$P(\omega_j|\mathbf{x}) = P(\omega_j)^{(L-1)} \prod_{i=1}^{L} P(\omega_j|\mathbf{x}^{(i)}) \times \frac{\prod_{i=1}^{L} p(\mathbf{x}^{(i)})}{p(\mathbf{x})} \tag{5.49}$$

The fraction at the end does not depend on the class label k, therefore we can ignore it when calculating the support $\mu_j(\mathbf{x})$ for class ω_j. Taking the classifier output $d_{i,k}(\mathbf{x}^{(i)})$ as the estimate of $P(\omega_j|\mathbf{x}^{(i)})$ and estimating the prior probabilities for the classes from the data, the support for ω_j is calculated as the (*weighted or probabilistic*) product combination rule

$$P(\omega_j|\mathbf{x}) \propto P(\omega_j)^{(L-1)} \prod_{i=1}^{L} P(\omega_j|\mathbf{x}^{(i)}) \tag{5.50}$$

$$= \hat{P}(\omega_j)^{(L-1)} \prod_{i=1}^{L} d_{i,k}(\mathbf{x}^{(i)}) = \mu_j(\mathbf{x}) \tag{5.51}$$

Kittler et al. [11] take this formula further to derive the sum combiner. Suppose that we have only the prior probabilities for the classes, $P(\omega_j)$, and no knowledge of $\mathbf{x}$. We can make a classification decision assigning always the most probable class, which we shall call a blind decision. Suppose that classifiers D_i, $i = 1, \ldots, L$, only slightly improve on the accuracy of the blind decision. In other words, the posterior probabilities differ only by a small fraction $\delta_{j,i}$ from the prior probabilities $P(\omega_j)$, $j = 1, \ldots, c$, where $|\delta_{j,i}| \ll 1$, that is,

$$P(\omega_j|\mathbf{x}^{(i)}) = P(\omega_j)(1 + \delta_{j,i}) \tag{5.52}$$

Substituting in Eq. (5.50),

$$P(\omega_j|\mathbf{x}) \propto P(\omega_j) \prod_{i=1}^{L} (1 + \delta_{j,i}) \tag{5.53}$$

Expanding the product and ignoring all terms of order two and higher with respect to $\delta_{j,i}$, we obtain

$$P(\omega_j|\mathbf{x}) \propto P(\omega_j)(1-L) + \sum_{i=1}^{L} P(\omega_j|\mathbf{x}^{(i)}) \tag{5.54}$$

The example below illustrates the derivation of Eq. (5.54) from Eq. (5.53).

Example: Illustration of the Derivation of the Sum Rule. Consider an ensemble of $L = 3$ classifiers. Denote by P the prior probability for class ω_j and by δ_1, δ_2, δ_3 the small deviations from P for the three classifiers, respectively. Processing the right-hand side of Eq. (5.53) by expanding the product and ignoring the terms of order two and higher, we obtain

$$P(\omega_j)\prod_{i=1}^{L}(1+\delta_{j,i}) \tag{5.55}$$

$$= P(1+\delta_1)(1+\delta_2)(1+\delta_3) \tag{5.56}$$

$$= P(1+\delta_1+\delta_2+\delta_3+\delta_1\delta_2+\delta_1\delta_3+\delta_2\delta_3+\delta_1\delta_2\delta_3) \tag{5.57}$$

$$\approx P[(1+\delta_1)+(1+\delta_2)+(1+\delta_3)-2] \tag{5.58}$$

$$= P(1+\delta_1)+P(1+\delta_2)+P(1+\delta_3)-2P \tag{5.59}$$

$$\propto \frac{1}{L}\left[P(\omega_j|\mathbf{x}^{(1)})+P(\omega_j|\mathbf{x}^{(2)})+P(\omega_j|\mathbf{x}^{(3)})\right]+\frac{(1-L)}{L}P(\omega_j) \tag{5.60}$$

$$\approx \frac{1}{L}\sum_{i=1}^{L} d_{i,k}(\mathbf{x}^{(i)})+\frac{(1-L)}{L}P(\omega_j) \tag{5.61}$$

which gives the average rule with a correction for the prior probability.

For equal prior probabilities Eqs. (5.51) and (5.54) reduce respectively to the most widely used variants of product (5.15) and mean (5.13). Kittler et al. investigate the error sensitivity of the product and sum combiners [11]. Their results show that the sum combiner is much more resilient to estimation errors of the posterior probabilities $P(\omega_j|\mathbf{x}^{(i)})$ than the product combiner. The product combiner is oversensitive to estimates close to zero. Presence of such estimates has the effect of veto on that particular class regardless of how large some of the estimates of other classifiers might be.

The authors find the good behavior of the average combiner surprising as this combiner has been derived under the strongest assumptions. Other studies derive the average combiner from different perspectives, which may explain its accuracy and robustness.

5.4.2 A Bayesian Perspective

Suppose there is one right classifier for the problem at hand, and by building an ensemble we are trying to approximate that classifier. This is called the *veridical*

assumption [169]. We sample from the set of possible models (called *hypotheses* in the machine learning literature) and estimate the predictions of these models. Let $\hat{P}(\omega_i|\mathbf{x}, D, \mathbf{Z})$ be the estimate of the posterior probability of class ω_i given $\mathbf{x}$, classifier D, and the labeled training set $\mathbf{Z}$. Denote by $P(D|\mathbf{Z})$ the probability that D is the right classifier for this problem, after having seen the data $\mathbf{Z}$. Given $\mathbf{x} \in \Re^n$, we can take the predicted probability to be the *expected* probability $\hat{P}(\omega_i|\mathbf{x}, D, \mathbf{Z})$ across all models D

$$\hat{P}(\omega_i|\mathbf{x}, \mathbf{Z}) = \sum_{D} \hat{P}(\omega_i|\mathbf{x}, D, \mathbf{Z})P(D|\mathbf{Z}) \tag{5.62}$$

If we take the expectation to be the average of the estimates given by the classifiers in the team ($P(D|\mathbf{Z}) = 1/L$), we arrive at the *average combiner* (the same as the mean and the sum combiner) (5.13).

We may argue that the probability of D_i being the right classifier for the problem, given the data set $\mathbf{Z}$, is measured by the estimated accuracy of D_i on $\mathbf{Z}$. Suppose that the set of possible classifiers is our ensemble $\mathcal{D}$ and we have an estimate of the probability that D_i is the correct classifier for the problem in terms of its performance on $\mathbf{Z}$. The probability $P(D_i|\mathbf{Z})$ will be the estimated accuracy of D_i normalized by dividing by the sum of all the L accuracies so that $P(D_i|\mathbf{Z})$, $i = 1, \ldots, L$, form a probability mass function of the population of classifiers $D_1, \ldots, D_L$.

Using the estimated accuracies as weights gives rise to the following version of the *weighted average combiner*:

$$\mu_j(\mathbf{x}) = \hat{P}(\omega_j|\mathbf{x}, \mathbf{Z}) \propto \sum_{i=1}^{L} [\hat{P}(D_i|\mathbf{Z})]\, d_{i,j}(\mathbf{x}) \tag{5.63}$$

A more precise but also more difficult to implement scenario is to assume that the probability for D_i being the right classifier depends on $\mathbf{x}$ [170]. Then $\hat{P}(D_i|\mathbf{Z})$ becomes $\hat{P}(D_i|\mathbf{x}, \mathbf{Z})$. Assigning data-specific weights underlies the mixture-of-experts (ME) model [169,191].

Another way to devise the coefficients in the weighted average is to follow the Bayesian approach a step further expanding $P(D_i|\mathbf{Z})$ as

$$P(D_i|\mathbf{Z}) = \frac{P(D_i)P(\mathbf{Z}|D_i)}{P(\mathbf{Z})} \tag{5.64}$$

where $P(D_i)$ expresses our belief that D_i is the right classifier for the problem, prior to seeing the data. The probability $P(\mathbf{Z}|D_i)$ can be interpreted as the likelihood of $\mathbf{Z}$ given that D_i is the right classifier for the problem. Since $\mathbf{Z}$ consists of independent identically distributed elements (iid) $\mathbf{z}_j$ with labels $l(\mathbf{z}_k)$, the likelihood of $\mathbf{Z}$ can be expressed as the product of the individual likelihoods

$$P(\mathbf{Z}|D_i) = \prod_{k=1}^{N} P(\mathbf{z}_k, l(\mathbf{z}_k)|D_i) \tag{5.65}$$

The pair $(\mathbf{z}_k, l(\mathbf{z}_k))$ depends on D_i only through the class label $l(\mathbf{z}_k)$, which implies $P(\mathbf{z}_k|D_i) = P(\mathbf{z}_k)$. Therefore

$$P(\mathbf{z}_k, l(\mathbf{z}_k)|D_i) = P(l(\mathbf{z}_k)|\mathbf{z}_k, D_i)P(\mathbf{z}_k|D_i) \tag{5.66}$$

$$= P(l(\mathbf{z}_k)|\mathbf{z}_k, D_i)P(\mathbf{z}_k) \tag{5.67}$$

and

$$P(\mathbf{Z}|D_i) = \prod_{k=1}^{N} P(l(\mathbf{z}_k)|\mathbf{z}_k, D_i)P(\mathbf{z}_k) = \prod_{k=1}^{N} P(l(\mathbf{z}_k)|\mathbf{z}_k, D_i) \prod_{k=1}^{N} P(\mathbf{z}_k) \tag{5.68}$$

Suppose that D_i has the same chance E_i to assign the wrong class label to any $\mathbf{x} \in \Re^n$. (Domingos calls this the *uniform class noise model* [192].) Then

$$P(l(\mathbf{z}_k)|\mathbf{z}_k, D_i) = \begin{cases} 1 - E_i, & \text{if } D_i \text{ guesses correctly the label of } \mathbf{z}_k, \\ E_i, & \text{otherwise.} \end{cases} \tag{5.69}$$

Then, substituting in Eq. (5.64),

$$P(D_i|\mathbf{Z}) = P(D_i)E_i^{N_E}(1 - E_i)^{(N-N_E)} \frac{\prod_{j=1}^{N} P(\mathbf{z}_j)}{P(\mathbf{Z})} \tag{5.70}$$

$$\propto P(D_i)E_i^{N_E}(1 - E_i)^{(N-N_E)} \tag{5.71}$$

where N_E is the number of points in $\mathbf{Z}$ misclassified by D_i. Assuming equal prior probabilities ($P(D_i) = 1/L$, $i = 1, \ldots, L$) and raising Eq. (5.71) to the power of $1/N$, we obtain

$$P(D_i|\mathbf{Z}) \propto \frac{1}{L} E_i^{N_E/N}(1 - E_i)^{(1-N_E/N)} \tag{5.72}$$

$$= \frac{1}{L} E_i^{E_i}(1 - E_i)^{(1-E_i)} \tag{5.73}$$

Here we assumed that E_i is approximated by N_E/N. Ignoring the constant $1/L$, which does not depend on the class label, and substituting in Eq. (5.63) leads to a new weighted average combiner

$$\mu_j(\mathbf{x}) = \hat{P}(\omega_j|\mathbf{x}, \mathbf{Z}) \propto \sum_{i=1}^{L} E_i^{E_i}(1 - E_i)^{(1-E_i)} d_{i,j}(\mathbf{x}) \tag{5.74}$$

Finally, we may decide to use the maximum likelihood method, and instead of averaging across D_i, pick the classifier with the lowest error rate E_i. Thus we dismiss

the ensemble and take the *single best* classifier. By doing so we risk to pick the classifier that only *appears* to be the best. The accuracy of this choice will depend on the accuracy of the estimates of E_i.

The above considerations show that there is no single theory underpinning classifier combination but a variety of explanations arising from different assumptions being made and different criteria being optimized. A summary of the simple and weighted combiners is given in Table 5.2.

Example: An Experiment with Weighted Averages on the Rotated Check-Board Data. The example in Section 5.2.1 is used again with the same experimental protocol. Previously we varied α in the generalized mean formula, achieving at best between 22 and 23 percent error. One hundred ensembles were generated again. The formulas from Table 5.2 were applied to each ensemble and the single best classifier was identified as the one with the lowest *training* error E_i. The results are shown in Figure 5.12.

The only prominent difference is between the single best classifier and the ensemble. The errors of the weighted averages are very similar, all in the range 22 to 23 percent. The simple (nonweighted) average was among the best competitors. We tried also the weighted average based on constrained regression (5.28) whose results on the previous example with the banana data were dramatically better than the simple average. For this problem though the differences were insignificant. This result reinforces the message that conclusions from a single experiment should not be taken as generally valid guidelines and recommendations for the choice of a combiner.

We can consider in the same framework the weighted average combiner with $c \times L$ weights (5.22), that is, where each class has a specific set of L weights.

TABLE 5.2 A Summary of Simple and Weighted Average Combiners with *L* Weights.

Name	Weights w_i	Comment
	$\mu_j(\boldsymbol{x}) = \sum_{i=1}^{L} w_i d_{i,j}(\boldsymbol{x})$	
Simple average (mean, sum rule)	1	The weights could be $1/L$. The constant does not affect the ordering of $\mu_j(\boldsymbol{x})$.
Weighted average 1	$1 - E_i$	Bayesian model: assume that $P(D_i\|\mathbf{Z})$ is the accuracy if D_i.
Weighted average 2	$E_i^{E_i}(1 - E_i)^{(1-E_i)}$	Bayesian model: uniform class noise model.
Weighted average 3	$1/E_i$	Derived from minimizing the ensemble error (Fumera and Roli [165], discussed in Chapter 9).

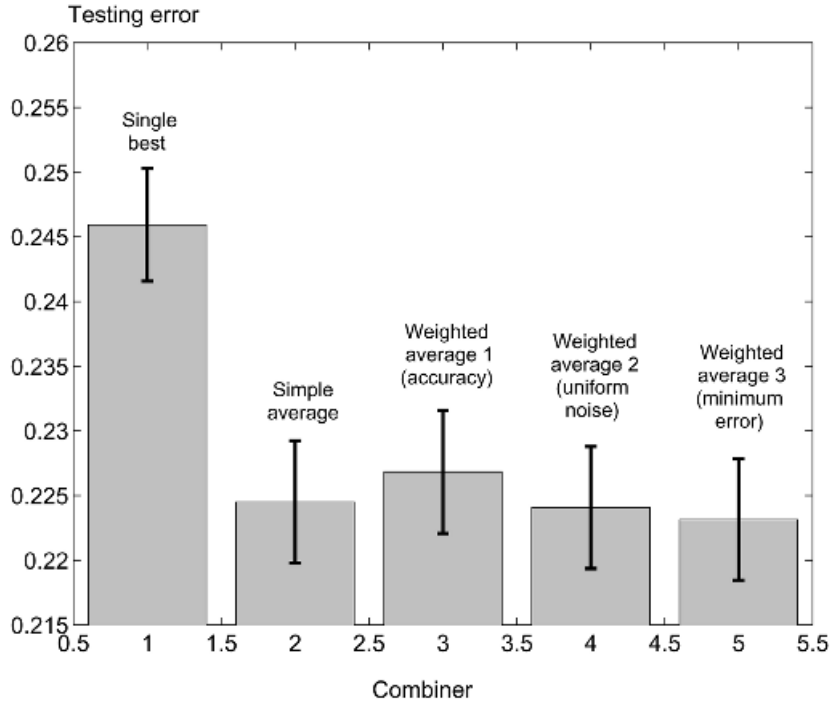


Fig. 5.12 *Results from simple and weighted average combinations averaged across 100 ensembles (of 10 Parzen classifiers each) on the rotated check-board data. The error bars mark the 95 percent CI for each result. The result for the single best classifier is also shown.*

5.4.3 The Supra Bayesian Approach

Jacobs [169] reviews methods for combining experts' probability assessments. *Supra Bayesian methods* consider the experts' estimates as "data" as many of the combiners do. The problem of estimating $\mu_j(\mathbf{x})$ becomes a problem of Bayesian learning in the intermediate feature space where the decision profile $DP(\mathbf{x})$ provides the $L \times c$ features. Loosely speaking, in the supra Bayesian approach for our task, we estimate the probabilities $\mu_j(\mathbf{x}) = P(\omega_j|\mathbf{x}), j = 1, \ldots, c$, using the L distributions provided by the ensemble members. Since these distributions are organized in a decision profile $DP(\mathbf{x})$, we have

$$\mu_j(\mathbf{x}) = P(\omega_j|\mathbf{x}) \propto p(DP(\mathbf{x})|\omega_j)P(\omega_j), \qquad j = 1, \ldots, c \tag{5.75}$$

where $p(DP(\mathbf{x})|\omega_j)$ is the joint class-conditional likelihood of the L classifier outputs, given $\mathbf{x}$ and ω_j (joint pdf). We assume that the only prior knowledge that we have is (some estimates of) the c prior probabilities $P(\omega_j)$.

When the classifier outputs are class labels, the supra Bayesian approach is equivalent to the multinomial combination method, called also BKS (Chapter 4). For continuous-valued outputs, this approach, albeit theoretically well motivated, is impractical [169]. The reason is that the pdf $p(DP(\mathbf{x})|\omega_j)$ is difficult to estimate. In principle, the supra Bayesian approach means that we use the intermediate feature space to build a classifier that is as close as possible to the Bayes classifier thereby guaranteeing the minimum possible classification error rate. Viewed in this light, all combiners that treat the classifier outputs in $DP(\mathbf{x})$ as new features are approximations within the supra Bayesian framework.

5.4.4 Kullback–Leibler Divergence

Miller and Yan [158] offer a theoretical framework for the average and product combiners based on the *Kullback–Leibler (K-L) divergence*. Kullback–Leibler divergence measures the distance between two probability distributions, p (prior distribution) and q (posterior distribution). It is also called "relative entropy" or "cross-entropy," denoted by $KL(p \parallel q)$. It can be interpreted as the amount of information necessary to change the prior probability distribution p into posterior probability distribution q. For a discrete x,

$$KL(p \parallel q) = \sum_x q(x) \log_2 \left(\frac{q(x)}{p(x)} \right) \tag{5.76}$$

It is assumed that for any x, if $p(x) = 0$ then $q(x) = 0$, and also $0 \times \log(0/p) = 0$ [193].

We regard each row of $DP(\mathbf{x})$ as a prior probability distribution on the set of class labels Ω and use $d_{i,j}(\mathbf{x})$ to denote the estimate of the probability $P(\omega_j|\mathbf{x}, D_i)$. Denote by $P_{(i)}$ the probability distribution on Ω provided by classifier D_i, that is, $P_{(i)} = (d_{i,1}(\mathbf{x}), \ldots, d_{i,c}(\mathbf{x}))$. For example, let $DP(\mathbf{x})$ be

$$DP(\mathbf{x}) = \begin{bmatrix} 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}$$

Then $P_{(1)} = (0.3, 0.7)$ is the pmf on $\Omega = \{\omega_1, \omega_2\}$ due to classifier D_1.

Given the L sets of probability estimates, our first hypothesis is that the true values of $P(\omega_i|\mathbf{x})$ (posterior probabilities) are the ones most agreed upon by the ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$. Denote by $P_{\text{ens}} = (\mu_1(\mathbf{x}), \ldots, \mu_c(\mathbf{x}))$ the probability mass function (pmf) over Ω with the guessed probabilities $\mu_j(\mathbf{x}) = P(\omega_j|\mathbf{x}, \mathcal{D})$. Then the averaged K-L divergence across the L ensemble members is

$$KL_{\text{av}} = \frac{1}{L} \sum_{i=1}^{L} KL(P_{\text{ens}} \parallel P_{(i)}) \tag{5.77}$$

We seek P_{ens} that minimizes Eq. (5.77). To simplify notation we shall drop the $(\mathbf{x})$ from $\mu_j(\mathbf{x})$ and $d_{i,j}(\mathbf{x})$ keeping in mind that we are operating on a specific point $\mathbf{x}$ in the feature space $\Re^n$. Take $\partial KL_{\text{av}}/\partial\mu_j$, include the term with the Lagrange multiplier to ensure that P_{ens} is a pmf, and set to zero

$$\frac{\partial}{\partial\mu_j}\left[KL_{\text{av}}+\lambda\left(1-\sum_{k=1}^{c}\mu_k\right)\right]=\frac{1}{L}\sum_{i=1}^{L}\frac{\partial}{\partial\mu_j}\left[\sum_{k=1}^{c}\mu_k\log_2\left(\frac{\mu_k}{d_{i,k}}\right)\right]-\lambda \tag{5.78}$$

$$=\frac{1}{L}\sum_{i=1}^{L}\left(\log_2\left(\frac{\mu_j}{d_{i,j}}\right)+C\right)-\lambda=0 \tag{5.79}$$

where $C=1/\ln(2)$. Solving for μ_j, we obtain

$$\mu_j=2^{(\lambda-C)}\prod_{i=1}^{L}(d_{i,j})^{1/L} \tag{5.80}$$

Substituting Eq. (5.80) in $\sum_{k=1}^{c}\mu_j=1$ and solving for λ gives

$$\lambda=C-\log_2\left(\sum_{k=1}^{c}\prod_{i=1}^{L}(d_{i,k})^{1/L}\right) \tag{5.81}$$

which back in Eq. (5.80) yields the final expression for the ensemble probability for class ω_j given the input $\mathbf{x}$ as the normalized geometric mean

$$\mu_j=\frac{\prod_{i=1}^{L}(d_{i,j})^{1/L}}{\sum_{k=1}^{c}\prod_{i=1}^{L}(d_{i,k})^{1/L}} \tag{5.82}$$

Notice that the denominator of μ_j does not depend on j. Also, the power $1/L$ in the numerator is only a monotone transformation of the product and will not change the ordering of the discriminant functions obtained through product. Therefore, the ensemble degree of support for class ω_j, $\mu_j(\mathbf{x})$ reduces to the *product combination rule*

$$\mu_j=\prod_{i=1}^{L}d_{i,j} \tag{5.83}$$

If we swap the places of the prior and posterior probabilities in Eq. (5.77) and again look for a minimum with respect to μ_j, we obtain

$$\frac{\partial}{\partial\mu_j}\left[KL_{\text{av}}+\lambda\left(1-\sum_{k=1}^{c}\mu_k\right)\right]=\frac{1}{L}\sum_{i=1}^{L}\frac{\partial}{\partial\mu_j}\left[\sum_{k=1}^{c}d_{i,k}\log_2\left(\frac{d_{i,k}}{\mu_k}\right)\right]-\lambda \tag{5.84}$$

$$=-\frac{1}{CL\mu_j}\sum_{i=1}^{L}d_{i,j}-\lambda=0 \tag{5.85}$$

where C is again $1/\ln(2)$. Solving for μ_j, we obtain

$$\mu_j = -\frac{1}{\lambda CL}\sum_{i=1}^{L} d_{i,j} \tag{5.86}$$

Substituting Eq. (5.86) in $\sum_{k=1}^{c}\mu_k = 1$ and solving for λ gives

$$\lambda = -\frac{1}{CL}\sum_{k=1}^{c}\sum_{i=1}^{L} d_{i,k} = -\frac{L}{CL} = -\frac{1}{C} \tag{5.87}$$

The final expression for the ensemble probability for class ω_j given the input $\mathbf{x}$ as the normalized arithmetic mean

$$\mu_j = \frac{1}{L}\sum_{i=1}^{L} d_{i,j} \tag{5.88}$$

which is the *mean combination rule* (the same as average or sum combiner).

The sum combiner was derived in the same way as the product combiner under a slightly different initial assumption. We assumed that P_{ens} is some unknown prior pmf that needs to be transformed into the L posterior pmfs suggested by the L ensemble members. Thus, to derive the average rule, we minimized the average information necessary to transform P_{ens} to the individual pmfs.

Miller and Yan go further and propose weights that depend on the "critic" for each classifier and each $\mathbf{x}$ [158]. The critic estimates the probability that the classifier is correct in labeling $\mathbf{x}$. Miller and Yan derive the product rule with the critic probability as the power of $d_{i,j}$ and the sum rule with the critic probabilities as weights. Their analysis and experimental results demonstrate the advantages of the weighted rules. The authors admit though that there is no reason why one of the set-ups should be preferred to the other.

5.4.5 Consensus Theory

Berenstein et al. [194] bring to the attention of the Artificial Intelligence community the so-called *consensus theory*, which has enjoyed a considerable interest in social and management sciences but remained not well known elsewhere. The theory looks into combining expert opinions and in particular combining L probability distributions on Ω (the decision profile $DP(\mathbf{x})$ in our case) into a single distribution $((\mu_1(\mathbf{x}), \ldots, \mu_c(\mathbf{x}))$ in our case). A *consensus rule* defines the way this combination is carried out. Consensus rules are derived so as to satisfy a set of desirable theoretical properties [188,195,196].

Based on an experimental study, Ng and Abramson [196] advocate using simple consensus rules such as the weighted average, called the *linear opinion pool* (5.21),

and the weighted product

$$\mu_j(\mathbf{x}) = \prod_{i=1}^{L} (d_{i,j}(\mathbf{x}))^{w_i} \tag{5.89}$$

called the *logarithmic opinion pool*. The approach taken to the weights assignment in consensus theory is based on the decision maker's knowledge of the importance of the experts (classifiers). In some weight assignment schemes the experts are assumed capable of assessing their own importance and also the importance of the other experts [194].

5.5 COMMENTS

1. The weighted average class of rules have been most widely used due to their simplicity and consistently good performance.
2. For some problems the simple average appears to be a surprisingly good rival to the weighted average combiners [157,159,160,196]. It is stable, although not always the most accurate combiner. An error-optimal weighted average with L weights has been found to be only marginally better than single average [165] (discussed in more detail in Chapter 9).
3. There has been a debate in the literature concerning weighted averages. The questions are whether
   - the weights should be positive;
   - the weights should be constrained to sum to one;
   - there should be an intercept term.

   The consensus at present is that neither of these options will make a significant difference to the final outcome [155,169]. Ting and Witten [155] advocate a model without an intercept and with nonnegative weights due to their interpretation potential.
4. The class-indifferent and class-conscious *weighted average* combiners, Eqs. (5.22) and (5.23), have been found to be approximately equivalent [155,171], therefore the simpler model with $c \times L$ weights is preferred (class-conscious).
5. Decision templates are perhaps the simplest class-indifferent combiner. "Decision templates" is a different name for the nearest mean classifier in the intermediate feature space. Using the whole $DP(\mathbf{x})$ gives a richer platform for building the combiner than using the L supports for each class separately. However, the issue of training of the combiner comes center stage. Stacked generalization is one possible solution where the combiner is trained on unseen data constructed by a cross-validation procedure. In modern applications the problem sometimes is the excessive amount of data rather than

the shortage of data. In such cases the system designer can afford to train the combiner on a separate validation set.

6. Based exclusively on empirical evidence, fuzzy integral has been advocated in several studies as a very successful combiner.
7. As always, there is no unique best combiner for all problems.

APPENDIX 5A CALCULATION OF λ FOR THE FUZZY INTEGRAL COMBINER

In the code below we assume that g is a vector (could be a row or a column vector) with the individual strengths of the L classifiers. First we transform Eq. (5.33) into $p(\lambda) = 0$ where

$$p(\lambda) \equiv \prod_{i=1}^{L} g^i \prod_{i=1}^{L} \left(\lambda - \left(-\frac{1}{g^i}\right)\right) - \lambda - 1$$

To find the coefficients we use the Matlab function `poly` to convert the roots $-(1/g^i)$ into a polynomial. The coefficients are multiplied by the product of the g^i terms. The last two coefficients are identified and decremented by 1 in order to take into account the terms "$-\lambda$" and "-1". The function `roots` finds all the roots of $p(\lambda) = 0$. The next line in the code identifies the index of λ and the value of λ is stored in `lambda`.

```
function lambda=lambda_fi(g)
gb=prod(g)*poly(-1./g); % find the coefficients of p(lambda)
gb(end-1:end)=gb(end-1:end)-1; % correct the last two
gc=roots(gb); % find the roots of p(lambda)=0
ii=(imag(gc)==0)&(gc>=-1)...
    &(abs(gc)>0.001); % identify lambda's index
lambda=(gc(ii)); % find lambda
```

6

Classifier Selection

6.1 PRELIMINARIES

The presumption in classifier selection is that there is an "oracle" that can identify the best expert for a particular input $\mathbf{x}$. This expert's decision is accepted as the decision of the ensemble for $\mathbf{x}$.

The idea of using different classifiers for different inputs was suggested by Dasarathy and Sheela back in 1978 [118]. They combine a linear classifier and a k-nearest neighbor (k-nn) classifier. The composite classifier identifies a conflict domain in the feature space and uses k-nn in that domain while the linear classifier is used elsewhere. In 1981 Rastrigin and Erenstein [119] gave a methodology for classifier selection almost in the form it is used now.

We may assume that the classifier "realizes" its competence for labeling $\mathbf{x}$. For example, if 10-nn is used and 9 of the 10 neighbors suggest the same class label, then the confidence in the decision is high. If the classifier outputs are reasonably well-calibrated estimates of the posterior probabilities, that is, $d_{i,j} = \hat{P}(\omega_j|\mathbf{x}, D_i)$, then the confidence of classifier $D_i \in \mathcal{D}$ for object $\mathbf{x}$ can be measured as

$$C(D_i|\mathbf{x}) = \max_{j=1}^{c} \hat{P}(\omega_j|\mathbf{x}, D_i) \tag{6.1}$$

where c is the number of classes.

Consider a cascade structure of classifiers. When the first classifier's confidence is high (above some predefined threshold) we take the class suggested by D_i as the

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

label for $\mathbf{x}$. If $C(D_i|\mathbf{x})$ is low, $\mathbf{x}$ is passed on to another classifier in the cascade which processes it in the same fashion. Finally, if the last classifier in the system is also uncertain, the system might refrain from making a decision or may select the most likely class anyway. Such cascade models are extremely useful for real-time systems where the majority of the inputs will only need to be processed by a few classifiers [111,197–201].

The recent interest in classifier selection was triggered by a publication by Woods et al. [120]. They introduced the term *dynamic classifier selection* to denote the process of choosing a member of the ensemble to make the decision based on the input $\mathbf{x}$.

To construct a classifier selection ensemble, the following questions need to be answered:

- How do we build the individual classifiers?
- How do we evaluate the competence of the classifiers for a given $\mathbf{x}$? If several classifiers tie as the most competent candidates, how do we break the tie?
- Once the competences are found, what selection strategy shall we use? The standard strategy is to select one most competent classifier and take its decision. However, if there are several classifiers of equally high competence, do we take one decision or shall we fuse the decisions of the most competent classifiers? When is it beneficial to select one classifier to label $\mathbf{x}$ and when should we be looking for a fused decision?

6.2 WHY CLASSIFIER SELECTION WORKS

We consider an ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$ of classifiers which have been already trained. Let $\Re^n$ be divided into K *selection regions* (called also *regions of competence*), $K > 1$. Denote the regions by $R_1, \ldots, R_K$. These regions are not associated with specific classes, nor do they need to be of a certain shape or size.

Example: Selection Regions. An example of partitioning into regions is shown in Figure 6.1. Depicted is a banana data set with 2000 data points. There are two classes and therefore *two classification regions*. Suppose that we have three classifiers: D_1, which always predicts ω_1; D_2, which always predicts ω_2; and a linear classifier D_3, whose discriminant function is shown as the thick dashed line in Figure 6.1. The individual accuracy of each of the three classifiers is approximately 0.5. A majority vote between them is also useless as it will always match the decision of D_3 and lead to 50 percent error as well. However, if we use the three selection regions and nominate one classifier for each region, the error of the ensemble will be negligible.

This example only shows the potential of the classifier selection approach. In practice we will hardly be so fortunate to find regions that will have such a dramatic effect on the ensemble performance. The main problem in classifier selection is exactly the *training* of the ensemble: finding the regions of competence, estimating

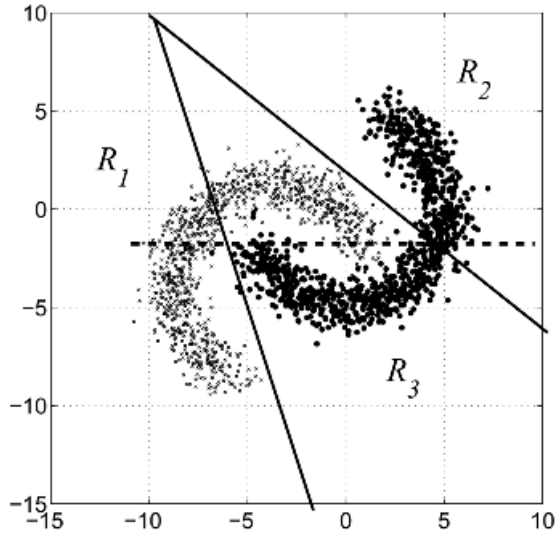


Fig. 6.1 *An example of partitioning the feature space with two classes into three selection regions.*

the competence, and choosing a selection mode (e.g., whether to nominate one single classifier or the best three, or weight the decisions by the competences, and so on).

Let D^* be the classifier with the highest average accuracy among the elements of $\mathcal{D}$ over the whole feature space $\Re^n$. Denote by $P(D_i|R_j)$ the probability of correct classification by D_i in region R_j. Let $D_{i(j)} \in \mathcal{D}$ be the classifier responsible for region $R_j, j = 1, \ldots, K$. The overall probability of correct classification of our classifier selection system is

$$P(correct) = \sum_{j=1}^{K} P(R_j)P(D_{i(j)}|R_j) \tag{6.2}$$

where $P(R_j)$ is the probability that an input $\mathbf{x}$ drawn from the distribution of the problem falls in R_j. To maximize $P(correct)$, we assign $D_{i(j)}$ so that

$$P(D_{i(j)}|R_j) \geq P(D_t|R_j), \forall\, t = 1, \ldots, L \tag{6.3}$$

Ties are broken randomly. From Eqs. (6.2) and (6.3),

$$P(correct) \geq \sum_{j=1}^{K} P(R_j)P(D^*|R_j) = P(D^*) \tag{6.4}$$

The above equation shows that the combined scheme is at least as good as the best classifier D^* in the pool $\mathcal{D}$, *regardless of the way the feature space has been partitioned* into selection regions. The only condition (and, of course, the trickiest one) is to ensure that $D_{i(j)}$ is indeed the best among the L classifiers from $\mathcal{D}$ for region R_j. The extent to which this is satisfied determines the success of the classifier selection model.

6.3 ESTIMATING LOCAL COMPETENCE DYNAMICALLY

In this approach the classifier chosen to label an input $\mathbf{x}$ is selected by estimating the local accuracy of the competitor classifiers *during the operational phase*.

6.3.1 Decision-Independent Estimates

In Ref. [202] this approach is called the *a priori* approach because the competence is determined based only on the location of $\mathbf{x}$, prior to finding out what labels the classifiers suggest for $\mathbf{x}$.

6.3.1.1 Direct k-nn Estimate. One way to estimate the competence is to identify the K nearest neighbors of $\mathbf{x}$ from either the training set or a validation set and find out how accurate the classifiers are on these K objects [119,120]. K is a parameter of the algorithm, which needs to be tuned prior to the operational phase.

6.3.1.2 Distance-Based k-nn Estimate. If the classifiers produce soft outputs, these can be used in the estimate. Giacinto and Rol [202] propose to estimate the competence of classifier D_i as the weighted average of the classifier's predictions for the *correct* labels of the K neighbors of $\mathbf{x}$. Denote by $P_i(l(\mathbf{z}_j)|\mathbf{z}_j)$ the estimate given by D_i of the probability for the true class label of $\mathbf{z}_j$. For example, suppose that the output of D_i for $\mathbf{z}_j$ in a five-class problem is [0.1, 0.4, 0.1, 0.1, 0.3]. Let the true class label of $\mathbf{z}_j$ be $l(\mathbf{z}_j) = \omega_5$. Although the decision of D_i would be for class ω_2, the suggested probability for the correct class label is $P_i(l(\mathbf{z}_j)|\mathbf{z}_j) = 0.3$. The probabilities are weighted by the distances to the K neighbors. Let $N_{\mathbf{x}}$ denote the set of the K nearest neighbors of $\mathbf{x}$. The competence of classifier D_i for $\mathbf{x}$ is

$$C(D_i|\mathbf{x}) = \frac{\sum_{\mathbf{z}_j \in N_{\mathbf{x}}} P_i(l(\mathbf{z}_j)|\mathbf{z}_j)(1/d(\mathbf{x}, \mathbf{z}_j)}{\sum_{\mathbf{z}_j \in N_{\mathbf{x}}} (1/d(\mathbf{x}, \mathbf{z}_j)}, \tag{6.5}$$

where $d(\mathbf{x}, \mathbf{z}_j)$ is the distance between $\mathbf{x}$ and its nearest neighbor $\mathbf{z}_j \in N_{\mathbf{x}}$ according to an appropriate distance measure chosen in advance.

6.3.1.3 Potential Functions Estimate. Rastrigin and Erenstein [119] also consider a distance-based competence coming from the so-called *potential functions* model. We regard the feature space as a field and assume that each point in the data

set contributes to the "potential" in $\mathbf{x}$. The potential for classifier D_i at $\mathbf{x}$ corresponds to its competence. The higher the potential, the higher the competence. The individual contribution of $\mathbf{z}_j \in \mathbf{Z}$ to $C(D_i|\mathbf{x})$ is

$$\phi(\mathbf{x}, \mathbf{z}_j) = \frac{g_{ij}}{1 + \alpha_{ij}(d(\mathbf{x}, \mathbf{z}_j))^2} \tag{6.6}$$

where

$$g_{ij} = \begin{cases} 1, & \text{if } D_i \text{ recognizes correctly } \mathbf{z}_j \in N_{\mathbf{x}}, \\ -1, & \text{otherwise,} \end{cases} \tag{6.7}$$

and α_{ij} is a parameter that weights the contribution of $\mathbf{z}_j$. In the simplest case, $\alpha_{ij} = \alpha$ for all $i = 1, \ldots, L$, and $j = 1, \ldots, N$.[24] The competence is calculated as

$$C(D_i|\mathbf{x}) = \sum_{\mathbf{z}_j \in \mathbf{Z}} \phi(\mathbf{x}, \mathbf{z}_j) \tag{6.8}$$

The direct k-nn estimate is a variant of the potential function approach with the following distance function

$$d(\mathbf{x}, \mathbf{z}_j) = \begin{cases} 1 \text{ (or any positive constant)}, & \text{if } \mathbf{z}_j \in N_{\mathbf{x}}, \\ 0, & \text{otherwise.} \end{cases} \tag{6.9}$$

It is not clear whether the distance-based versions are better than the direct k-nn estimate of the competence. One advantage though is that ties are less likely to occur for the distance-based estimates.

6.3.2 Decision-Dependent Estimates

This approach is termed *a posteriori* in Ref. [202]. The class predictions for $\mathbf{x}$ by all the classifiers are known.

6.3.2.1 Direct k-nn Estimate. Let $s_i \in \Omega$ be the class label assigned to $\mathbf{x}$ by classifier D_i. Denote by $N_{\mathbf{x}}^{(s_i)}$ the set of K nearest neighbors of $\mathbf{x}$ from $\mathbf{Z}$, which classifier D_i labeled as s_i. The competence of classifier D_i for the given $\mathbf{x}$, $C(D_i|\mathbf{x})$, is calculated as the proportion of elements of $N_{\mathbf{x}}^{(s_i)}$ whose true class label was s_i. This estimate is called in Ref. [120] the *local class accuracy*.

6.3.2.2 Distance-Based k-nn Estimate. Denote by $P_i(s_i|\mathbf{z}_j)$ the estimate given by D_i of the probability that the true class label of $\mathbf{z}_j$ is s_i. The competence of D_i can be measured by $P_i(s_i|\mathbf{z}_j)$ averaged across the data points in the vicinity of $\mathbf{x}$ whose true label was s_i. Using the distances to $\mathbf{x}$ as weights, we calculate the

[24] Recall that y_{ij} was defined as the oracle output of classifier D_i for data point $\mathbf{z}_j$ (4.1). y_{ij} is 1 if D_i assigns the correct label and 0 otherwise. Note that $g_{ij} = 2 \times y_{ij} - 1$.

competence of D_i as

$$C(D_i|\mathbf{x}) = \frac{\sum_{\mathbf{z}_j} P_i(s_i|\mathbf{z}_j) 1/d(\mathbf{x}, \mathbf{z}_j)}{\sum_{\mathbf{z}_j} 1/d(\mathbf{x}, \mathbf{z}_j)} \tag{6.10}$$

where the summation is on $\mathbf{z}_j \in N_{\mathbf{x}}$ such that $l(\mathbf{z}_j) = s_i$. A different number of neighbors K can be considered for $N_{\mathbf{x}}$ and $N_{\mathbf{x}}^{(s_i)}$. Woods et al. [120] found the direct decision-dependent k-nn estimate of competence superior to that provided by the decision-independent estimate. They recommend $K = 10$ for determining the set $N_{\mathbf{x}}^{(s_i)}$.

Example: Estimation of Local Competence of Classifiers. The table below gives the true class labels and the guessed class labels using classifier D_i for a hypothetical set of 15 nearest neighbors of $\mathbf{x}$, $N_{\mathbf{x}}$. (To save space, only the indices of the objects and the classes are given.)

Object ($\mathbf{z}_j$)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
True label ($l(\mathbf{z}_j)$)	2	1	2	2	3	1	2	1	3	3	2	1	2	2	1
Guessed label (s_i)	2	3	2	2	1	1	2	2	3	3	1	2	2	2	1

The decision-independent direct k-nn estimate of competence of D_i is the accuracy of D_i calculated on $N_{\mathbf{x}}$, that is, $C(D_i|\mathbf{x}) = \frac{2}{3}$. Suppose that the output suggested by D_i for $\mathbf{x}$ is ω_2. The decision-dependent direct k-nn estimate of competence of D_i for $K = 5$ is the accuracy of D_i calculated on $N_{\mathbf{x}}^{\omega_2}$ where only the five nearest neighbors labeled in ω_2 are considered. Then $N_{\mathbf{x}}^{\omega_2}$ consists of objects $\{1, 3, 4, 7, 8\}$. As four of the elements of $N_{\mathbf{x}}^{\omega_2}$ have true labels ω_2, the local competence of D_i is $C(D_i|\mathbf{x}) = \frac{4}{5}$.

To illustrate the calculation of the distance-based estimates, we will assume the following:

- The indices of $\mathbf{z}_j$ in the first row in the table are the actual values of $\mathbf{z}_j \in \Re$ and $\mathbf{x} \in \Re$ is located at 0.
- Euclidean distance is used, for example, $d(\mathbf{x}, \mathbf{z}_3) = 3$.
- D_i provides only the label outputs. We associate probabilities $P_i(s_i|\mathbf{z}_j) = 1$ (the probability for the chosen output) and $P_i(s|\mathbf{z}_j) = 0$ for any other $s \in \Omega$, $s \neq s_i$. Thus, if the suggested class label is ω_2, then the "soft" vector with the suggested probabilities will be [0, 1, 0].

The decision-independent distance-based k-nn estimate of the competence of D_i (using the whole of $N_{\mathbf{x}}$) will be

$$C(D_i|\mathbf{x}) = \frac{1 + \frac{1}{3} + \frac{1}{4} + \frac{1}{6} + \frac{1}{7} + \frac{1}{9} + \frac{1}{10} + \frac{1}{13} + \frac{1}{14} + \frac{1}{15}}{1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{15}} \approx 0.70$$

The potential function estimate is calculated in a similar way.

Let again the output suggested by D_i for $\mathbf{x}$ be ω_2. For the decision-dependent estimate, using the whole of $N_{\mathbf{x}}$ and taking only the elements whose true label was ω_2,

$$C(D_i|\mathbf{x}) = \frac{1+\frac{1}{3}+\frac{1}{4}+\frac{1}{7}+\frac{1}{13}+\frac{1}{14}}{1+\frac{1}{3}+\frac{1}{4}+\frac{1}{7}+\frac{1}{11}+\frac{1}{13}+\frac{1}{14}} \approx 0.95$$

6.3.3 Tie-Break for Classifiers with Equal Competences

The tie-break within the dynamic classifier selection model [120] is carried out as follows:

1. Upon obtaining an input $\mathbf{x}$, label it by $D_1, \ldots, D_L$. If all classifiers agree on the label, then assign this label to $\mathbf{x}$ and return.
2. If there is a disagreement, then estimate the local accuracy, $C(D_i|\mathbf{x})$, for each D_i, $i = 1, \ldots, L$. To do this, take the class label s_i offered for $\mathbf{x}$ by D_i and find the K points closest to $\mathbf{x}$ for which D_i has issued the same label. Calculate the proportion of the points whose true labels were s_i to be an estimate of the local accuracy of D_i with respect to class s_i (decision-based direct k-nn estimate).
3. If there is a unique winner of the local accuracy contest, let it label $\mathbf{x}$ and return. Otherwise, check if the tied winners have the same labels for $\mathbf{x}$. If so, accept the label and return. If a unique class label could be selected by plurality among the tied classifiers, then assign this label to $\mathbf{x}$ and return.
4. Otherwise, there is a class label tie between the most locally competent classifiers. The classifier with the next highest local competence is identified to break the tie. If all classifiers are tied (there is no classifier left to break the tie), then pick a random class label among the tied *labels* and return. If there is a unique winner of the (second) local competence contest, and it can resolve the tie, then use the winning label for $\mathbf{x}$ and return.
5. If none of the clauses in the previous point applies, break the class label tie randomly and return a label for $\mathbf{x}$.

The last item in the list did not exist in the original tie-break procedure; we added it to break the recursion. Further analysis of the ties would have produced an overcomplicated code because there could be another tie on the second highest competence, where the tied classifiers disagree on the class label, and so on. Table 6.1 shows examples of cases of tie-breaks and the respective output labels.

The last case seems counterintuitive. There are many relatively competent classifiers which suggest label 2, yet we have chosen randomly between labels 1 and 2 proposed by the two most competent classifiers. The random choice was made because there was no single second-most-competent classifier. We could have gone further and take the majority vote between the four second-most-competent classifiers thereby supporting the vote for class 2. However, if there is a tie again, the analysis would become unnecessarily complicated and time-consuming. This is why item 5 above was introduced.

TABLE 6.1 Tie-Break Examples for the Dynamic Classifier Selection Model for an Ensemble of Nine Classifiers.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
Class labels	1	2	1	2	3	2	3	1	2
Competences	0.8	0.7	0.7	0.6	0.6	0.6	0.4	0.3	0.2
Final label: **1** (single most competent classifier)									
Class labels	1	2	2	2	3	2	3	1	2
Competences	0.7	0.7	0.7	0.6	0.6	0.6	0.4	0.3	0.2
Final label: **2** (majority of the competence-tied classifiers)									
Class labels	1	2	2	2	3	2	3	1	2
Competences	0.7	0.7	0.6	0.5	0.5	0.5	0.4	0.3	0.2
Final label: **2** (competence-tie resolved by the second most competent classifier)									
Class labels	1	2	1	1	2	2	3	1	2
Competences	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
Final label: **1** (random selection between competence-tied labels)									
Class labels	1	2	2	2	3	2	3	1	2
Competences	0.7	0.7	0.6	0.6	0.6	0.6	0.4	0.3	0.2
Final label: **1** (random selection between competence tied-labels (item 5))									

6.4 PREESTIMATION OF THE COMPETENCE REGIONS

Estimating the competence dynamically might be too computationally demanding. First, the K nearest neighbors of $\mathbf{x}$ have to be found. For the decision-dependent estimates of competence, $L \times K$ neighbors might be needed. Secondly, the estimates of the competence have to be calculated. To decrease the computational complexity, competences can be precalculated across *regions of competence*. When $\mathbf{x}$ is submitted for classification, the region of competence in which $\mathbf{x}$ falls is identified and $\mathbf{x}$ is labeled by the classifier responsible for that region. The problem becomes one of identifying the regions of competence and the corresponding classifiers.

The regions of competence can be chosen arbitrary as long as we have reliable estimates of the competences within these regions. Denote by K the number of regions of competence. The number of classifiers L is not necessarily equal to the number of regions K. Next we decide which classifier from $\mathcal{D} = \{D_1, \ldots, D_L\}$ should be picked for each region R_j, $j = 1, \ldots, K$. Some classifiers might never be nominated and therefore they are not needed in the operation of the combination scheme. Even the classifier with the highest accuracy over the whole feature space might be dropped from the final set of classifiers. On the other hand, one classifier might be nominated for more than one region.

The feature space can be split into regular regions but in this case some of the regions might contain only a small amount of data points and lead to spurious estimates of the competences. To ensure that the regions contain a sufficient amount of points we can use clustering and relate to each cluster a region of competence. The classifier whose accuracy is estimated to be the highest within a region R will be assigned to make the decisions for any future $\mathbf{x} \in R$.

Clustering and selection (training)

1. Design the individual classifiers $D_1, \ldots, D_L$ using the labeled data set $\mathbf{Z}$. Pick the number of regions K.
2. Disregarding the class labels, cluster $\mathbf{Z}$ into K clusters, $C_1, \ldots, C_K$, using, e.g., the K-means clustering procedure [2]. Find the cluster centroids $\mathbf{v}_1, \ldots, \mathbf{v}_K$ as the arithmetic means of the points in the respective clusters.
3. For each cluster C_j, (defining region R_j), estimate the classification accuracy of $D_1, \ldots, D_L$. Pick the most competent classifier for R_j and denote it by $D_{i(j)}$.
4. Return $\mathbf{v}_1, \ldots, \mathbf{v}_K$ and $D_{i(1)}, \ldots, D_{i(K)}$.

Fig. 6.2 *Training of the clustering and selection method.*

Clustering and selection (operation)

1. Given the input $\mathbf{x} \in \Re^n$, find the nearest cluster center from $\mathbf{v}_1, \ldots, \mathbf{v}_K$, say, $\mathbf{v}_j$.
2. Use $D_{i(j)}$ to label $\mathbf{x}$.

Fig. 6.3 *Operation of the clustering and selection method.*

6.4.1 Clustering

A classifier selection method called *clustering and selection* is proposed in Refs. [203, 204]. Figure 6.2 shows the training, and Figure 6.3 the operation of clustering and selection.

The regions $R_1, \ldots, R_K$ can be estimated prior to training any classifier. A classifier can be trained on the data for each region R_j to become the expert $D_{i(j)}$. The operational phase will be the same as shown in Figure 6.3. The performance of such a strategy will depend more strongly on the way the regions are assigned compared to the strategy whereby the competences are estimated across the regions.

6.4.2 Selective Clustering

Liu and Yuan propose a more selective clustering approach [205]. Instead of a single partition of the feature space, they consider one partition per classifier. Consider classifier D_i trained on the data set $\mathbf{Z}$.[25]

[25] As usual, a separate validation set for estimating regions and competences is recommended. To keep the notation simple, we shall use only one data set, $\mathbf{Z}$.

In the training phase, the data is separated into $\mathbf{Z}^+$ and $\mathbf{Z}^-$, such that $\mathbf{Z}^+$ contains only the objects correctly classified by D_i and $\mathbf{Z}^-$ contains the objects misclassified by $D_i(\mathbf{Z}^+ \cup \mathbf{Z}^- = \mathbf{Z})$. It is assumed that there are c hyper-elliptical clusters in $\mathbf{Z}^+$, each cluster corresponding to a class. The means and the covariance matrices of the clusters are estimated from the respective points that were correctly classified by D_i. In $\mathbf{Z}^-$, the number of clusters, K_i, is evaluated during the clustering procedure. The means and the covariance matrices of the new clusters are estimated from the respective data points. Thus each classifier in the ensemble suggests a partition of the feature space into $c + K_i$ clusters and produces the means and the covariance matrices of the suggested clusters. To complete the training, we estimate the competence of D_i in each of the $c + K_i$ regions suggested by it.

In the operational phase, the input to be classified, $\mathbf{x}$, is considered separately by each classifier, for example, D_i. The region of competence where $\mathbf{x}$ belongs is identified among the $c + K_i$ regions for D_i using the Mahalanobis distance from $\mathbf{x}$ to the cluster centers. $\mathbf{x}$ is placed in the region with the closest center. The competence of D_i for the region of $\mathbf{x}$ is retrieved. The competences of all L classifiers for the given $\mathbf{x}$ are calculated in this way and compared. The classifier with the highest competence gives the label of $\mathbf{x}$.

Liu and Yuan report superior results to these of the simple clustering and selection. Indeed, if we split the feature space into smaller regions, we have a more flexible model for labeling $\mathbf{x}$ but the estimates of the competences have to be reliable enough.

6.5 SELECTION OR FUSION?

Selection is guaranteed by design to give at least the same training accuracy as the best individual classifier D^*. However, the model might overtrain, giving a deceptively low training error. To guard against overtraining we may use confidence intervals and nominate a classifier only when it is significantly better than the others. A statistical test is performed for determining whether the best classifier in R_j, $D_{i(j)}$, is significantly different from the remaining lot. Since we are interested only in a difference between the best classifier and the rest, we can perform a pairwise test. It is enough to eliminate the second best classifier. If $D_{i(j)}$ is significantly better than the second best, then $D_{i(j)}$ can be nominated as the classifier responsible for region R_j. Otherwise, a scheme involving more than one classifier might pay off.

As an example, assume that five classifiers have been designed on a data set with 100 elements. Define

$$\mathbf{y}_i = [y_{i,1}, \ldots, y_{i,100}]^T \tag{6.11}$$

to be a vector with classification outcome of classifier D_i on the data set, such that $y_{ij} = 1$, if D_i recognizes correctly the jth element of the data set, and 0, otherwise. Table 6.2 shows the distribution of $\{\mathbf{y}_1, \ldots, \mathbf{y}_5\}$ for the 100 elements. The total number of correctly recognized objects is shown in the bottom row for each classifier. We could be tempted to nominate D_1 for region R_j as its classification accuracy is

TABLE 6.2 Distribution of Correct/Incorrect Classification Decisions for Five Classifiers for a Data Set with 100 Elements (Note that Not All Possible Combinations Have Occurred). The Bottom Row Contains the Total Number of Correctly Recognized Objects for Each Classifier.

y_1	y_2	y_3	y_4	y_5	Number
1	1	1	1	1	42
0	0	0	1	1	18
1	1	0	0	0	13
1	0	0	1	0	11
1	0	1	0	0	10
0	0	0	0	1	6
76	55	52	71	66	–

76 percent, higher by 5 percent than the second best. However, the paired t-test analysis suggests that D_1 is not significantly different from $D_4 (p = 0.438)$, not even from D_5 ($p = 0.191$). Therefore, it is probably better to consider a fusion scheme of the decisions of more than one classifier in R_j.

Figure 6.4 shows the 95 percent confidence intervals (CIs) of the classification accuracies of the five classifiers. The intervals have been calculated through the standard formula

$$\left[\hat{P}_D - t_{(0.05,N-1)} \sqrt{\frac{\hat{P}_D(1-\hat{P}_D)}{N}}, \hat{P}_D + t_{(0.05,N-1)} \sqrt{\frac{\hat{P}_D(1-\hat{P}_D)}{N}} \right] \tag{6.12}$$

where N is the sample size, $t_{(0.05,N-1)}$ is the t value for 95 percent significance level ($\alpha = 0.05$) and $N - 1$ degrees of freedom, and $\hat{P}_D$ is the estimate of the classification accuracy of the respective D_i in region R_j. For $N \geq 30$ we can use $t_{(0.05,N-1)} = 1.96$.

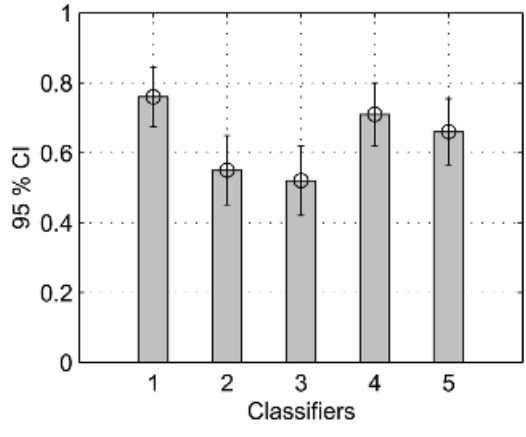


Fig. 6.4 *95 percent confidence intervals (CI) for the five classifiers.*

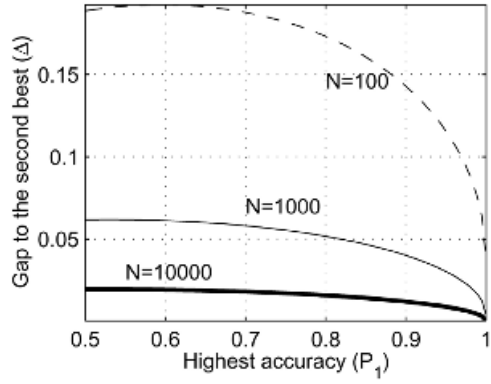


Fig. 6.5 *The difference Δ between the best and the second best classification accuracies in region R_j guaranteeing that the 95 percent CI of the two do not overlap.*

The above calculations are based on the assumption that $P(D_i|R_j)$ is the same for all $\mathbf{x} \in R_j$ and has therefore a Binomial distribution. We can derive an equation for calculating the "gap" needed so that the 95 percent CI of the best and the second best classifiers do not overlap. In this case the best classifier can be nominated as $D_{i(j)}$. Let $\Delta = P_1 - P_2$, $\Delta > 0$, where P_1 is the highest accuracy and P_2 is the second highest in R_j. The critical value for the gap Δ is derived from the equation

$$1.96\sqrt{\frac{P_1(1-P_1)}{N}} + 1.96\sqrt{\frac{P_2(1-P_2)}{N}} = \Delta \tag{6.13}$$

Substituting $P_2 = P_1 - \Delta$ and solving for Δ, we obtain

$$\Delta = \frac{7.6832\ P_1 - 3.8416 + 3.92\sqrt{NP_1(1-P_1)}}{N + 3.8416} \tag{6.14}$$

Figure 6.5 plots Δ against P_1 for three values of N: 100, 1000, and 10,000. For larger N the required gap for the best and the second best classifiers to be significantly different is smaller. In other words, even a 2 percent difference in the classification accuracy will suffice to nominate $D_{i(j)}$ as the classifier responsible for region R_j.

6.6 BASE CLASSIFIERS AND MIXTURE OF EXPERTS

Any set of classifiers might be suitable for the classifier selection model. Curiously, the experimental results with the dynamic classifier selection model reported by

Woods et al. [120] hardly improved upon the best classifier in the ensemble, which was often the nearest neighbor classifier. Nonetheless, the proposed dynamic classifier selection method was found by other authors believable and appealing even without convincing empirical support. Some authors suggest using bagging or boosting to develop the ensemble and use a selection strategy for combining [206,207].

An interesting ensemble method which belongs in the classifier selection group is the so called *mixture of experts* (ME) [191,200,209,210]. As illustrated in Figure 6.6, in this model the selector makes use of a separate classifier, which determines the participation of the experts in the final decision for an input $\mathbf{x}$. The ME architecture has been proposed for neural networks. The experts are neural networks, which are trained so that each NN is responsible for a part of the feature space. The selector uses the output of another neural network called the *gating network*. The input to the gating network is $\mathbf{x}$ and the output is a set of coefficients $p_1(\mathbf{x}), \ldots, p_L(\mathbf{x})$. Typically, $\sum_{i=1}^{L} p_i(\mathbf{x}) = 1$, and $p_i(\mathbf{x})$ is interpreted as the probability that expert D_i is the most competent expert to label the particular input $\mathbf{x}$. The probabilities are used together with the classifier outputs in one of the following ways:

- *Stochastic selection.* The classifier to label $\mathbf{x}$ is chosen by sampling from $\mathcal{D} = \{D_1, \ldots, D_L\}$ according to the distribution $p_1(\mathbf{x}), \ldots, p_L(\mathbf{x})$.
- *Winner-takes-all.* The classifier to label $\mathbf{x}$ is chosen by the maximum of $p_i(\mathbf{x})$.

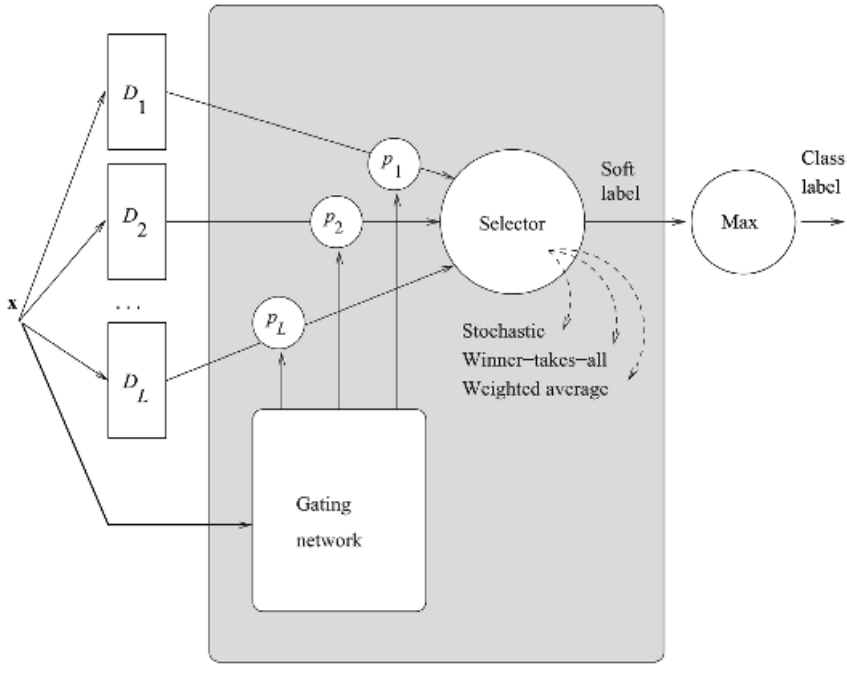


Fig. 6.6 *Mixture of experts.*

- *Weights*. The probabilities are used as the weighting coefficients to the classifier outputs. For example, suppose that the classifiers produce soft outputs for the c classes, $d_{i,j}(\mathbf{x}) \in [0, 1]$, $i = 1, \ldots, L$, $j = 1, \ldots, c$. Then the final (soft) output for class ω_j is

$$\mu_j(\mathbf{x}) = \sum_{i=1}^{L} p_i(\mathbf{x}) d_{i,j}(\mathbf{x}) \tag{6.15}$$

We note that the calculation of the final degree of support is the weighted average but in the ME model the coefficients depend on the input $\mathbf{x}$. If $d_{i,j}(\mathbf{x})$ are interpreted as probabilities, then $p_i(\mathbf{x})$ can be thought of as mixing proportions in estimating $\mu_j(\mathbf{x})$ by a mixture of distributions.

The most important question is how to train the ME model. Two training procedures are suggested in the literature. The first procedure is the standard error backpropagation implementing a *gradient descent*. According to this procedure, the training of an ME is no different in principle to training a single neural network with a complicated structure. The second training approach is based on the *expectation maximization* method, which appears to be faster than the gradient descent [209]. The mixture of experts method has been designed mainly for function approximation rather than classification. Its key importance for multiple classifier systems is the idea of training the gating network (therefore the selector) together with the individual classifiers through a standardized training protocol.

7

Bagging and Boosting

7.1 BAGGING

7.1.1 Origins: Bagging Predictors

Breiman introduced the term *bagging* as an acronym for *Bootstrap AGGregatING* [210]. The idea of bagging is simple and appealing: the ensemble is made of classifiers built on bootstrap replicates of the training set. The classifier outputs are combined by the plurality vote [18].

The diversity necessary to make the ensemble work is created by using different training sets. Ideally, the training sets should be generated randomly from the distribution of the problem. In practice, we can only afford one labelled training set, $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, and have to imitate the process or random generation of L training sets. We sample *with replacement* from the original training set (bootstrap sampling [212]) to create a new training set of length N. To make use of the variations of the training set, the base classifier should be *unstable*, that is, small changes in the training set should lead to large changes in the classifier output. Otherwise, the resultant ensemble will be a collection of almost identical classifiers, therefore unlikely to improve on a single classifier's performance. Examples of unstable classifiers are neural networks and decision trees while k-nearest neighbour is an example of a stable classifier. Figure 7.1 shows the training and operation of bagging.

Bagging is a parallel algorithm in both its training and operational phases. The L ensemble members can be trained on different processors if needed.

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

BAGGING

Training phase

1. Initialize the parameters
   - $\mathcal{D} = \emptyset$, the ensemble.
   - L, the number of classifiers to train.
2. For $k = 1, \ldots, L$
   - Take a bootstrap sample S_k from $\mathbf{Z}$.
   - Build a classifier D_k using S_k as the training set.
   - Add the classifier to the current ensemble, $\mathcal{D} = \mathcal{D} \cup D_k$.
3. Return $\mathcal{D}$.

Classification phase

4. Run $D_1, \ldots, D_L$ on the input $\mathbf{x}$.
5. The class with the maximum number of votes is chosen as the label for $\mathbf{x}$.

Fig. 7.1 *The bagging algorithm.*

7.1.2 Why Does Bagging Work?

If classifier outputs were independent and classifiers had the same individual accuracy p, then the majority vote is guaranteed to improve on the individual performance [131]. Bagging aims at developing independent classifiers by taking bootstrap replicates as the training sets. The samples are pseudo-independent because they are taken from the same $\mathbf{Z}$. However, even if they were drawn independently from the distribution of the problem, the *classifiers* built on these training sets might not give independent outputs.

Example: Independent Samples, Bootstrap Samples, and Bagging. The data for this example was the rotated check-board data set shown in Figure 1.10 in Chapter 1. A training set of 100 points and a testing set of 1000 points were generated 100 times. Bagging was run with decision tree as the base classifier. The trees were prepruned using a fixed threshold $\theta = 3$. To evaluate the effect of bootstrap sampling we ran the same experiment but instead of bootstrap samples of the training set, we generated a new training set of 100 objects for each new member of the ensemble.

Since there are only two classes, we represent one of the classes by 0 and the other by 1, and calculate the correlation between the outputs of classifiers D_i and D_j as

$$\rho_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11} + N^{10})(N^{01} + N^{00})(N^{11} + N^{01})(N^{10} + N^{00})}} \tag{7.1}$$

where N^{ab} is the number of objects in the testing set for which classifier D_i gives output a and classifier D_j gives output b, where $a, b \in \{0, 1\}$. To obtain a single value measuring the correlation for the whole ensemble, the pairwise correlations were averaged.

Figure 7.2*a* shows the average correlation between the ensemble members versus the ensemble size. When bagging is applied, the correlation is higher than when the training sets for the ensemble members are sampled directly from the distribution of the problem. The testing error rates versus the ensemble size are shown in Figure 7.2*b*.

The plots show that, as expected, the correlation between classifiers built on true independent samples is lower than that produced by boosting. Note, however, that the correlation for the true independent samples is not zero, which demonstrates that the outputs of classifiers built on independent samples might be dependent.

We can think of the ensembles with lower correlation as more diverse than those with higher correlation. Figure 7.2*b* shows the benefit of having more diverse ensembles. The base classifiers are the same, the pruning method is the same, the combiner is the same (majority vote) but the error rates are different. The improved error rate can therefore be attributed to higher diversity in the ensemble that uses independently generated training sets. We shall see later that the concept of diversity is not as simple and straightforward as it looks here.

Note that even though ensemble diversity does not increase with the ensemble size L, bagging keeps driving the error down. The individual accuracy of the ensemble members is not likely to change with L either, because all classifiers solve approximately the same problem. Therefore, the logical explanation of the success of bagging is the effect of the majority vote combiner. The bias-variance decomposition of the classification error, explained later, provides a (somewhat controversial) tool for analysis of this effect.

Domingos [192] examines two hypotheses about bagging in the framework of Bayesian learning theory. The first hypothesis is that bagging manages to estimate the posterior probabilities for the classes $\hat{P}(\omega_i|\mathbf{x})$ by conforming very well with the Bayesian model given by Eq. (5.62) as explained in Chapter 5. According to this model, the estimated posterior probability that the class label for the given $\mathbf{x}$ is ω_i, given the training set $\mathbf{Z}$, is averaged across all classifiers. Using $d_{j,i}(\mathbf{x})$ to

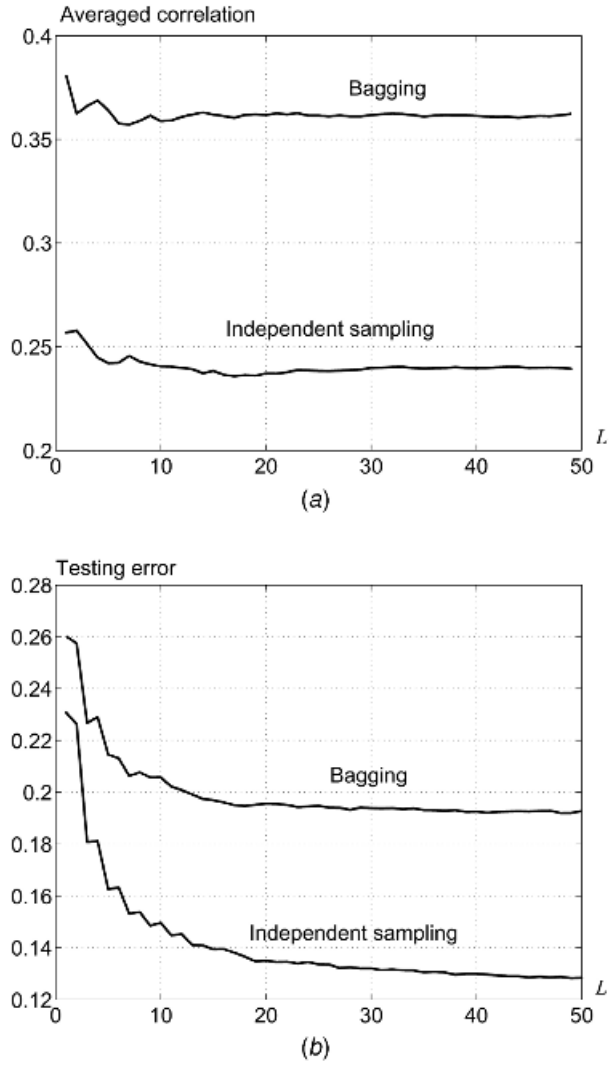


Fig. 7.2 *Bagging for the rotated check-board data using bootstrap samples and independent samples:* (*a*) *averaged pairwise correlation versus the ensemble size;* (*b*) *testing error rate versus the ensemble size.*

denote the estimate of $P(\omega_i|\mathbf{x})$ given by classifier D_j, we have

$$\hat{P}(\omega_i|\mathbf{x}, \mathbf{Z}) = \sum_{j=1}^{L} \hat{P}(\omega_i|\mathbf{x}, D_j, \mathbf{Z})\hat{P}(D_j|\mathbf{Z}) = \sum_{j=1}^{L} d_{j,i}(\mathbf{x})\hat{P}(D_j|\mathbf{Z}) \tag{7.2}$$

Take $d_{j,i}(\mathbf{x})$ to be the zero–one output indicating a (hard) class label. For example, in a four-class problem, a classifier output ω_3 corresponds to $d_{j,3}(\mathbf{x}) = 1$

and $d_{j,1}(\mathbf{x}) = d_{j,2}(\mathbf{x}) = d_{j,4}(\mathbf{x}) = 0$. If we set all the model (posterior) probabilities $P(D|\mathbf{Z})$ to $1/L$, we obtain the plurality voting combination, which is the traditional combiner for bagging. Domingos also looks at soft class labels and chooses $\hat{P}(D_j|\mathbf{Z})$ so that the combiner is equivalent to simple averaging and weighted averaging. The hypothesis that bagging develops a better estimate of the posterior probabilities within this Bayesian context was not supported by Domingos' experiments [192].

The second hypothesis obtained better empirical support. According to it, bagging shifts the prior distribution of the classifier models towards models that have higher complexity (as the ensemble itself). Such models are assigned a larger likelihood of being the right model for the problem. The ensemble is in fact a single (complex) classifier picked from the new distribution.

Domingos' conclusions are matched by an argument in Ref. [213] where the authors challenge the common intuition that voting methods work because they smooth out the estimates. They advocate the thesis that voting in fact increases the complexity of the system.

7.1.3 Variants of Bagging

7.1.3.1 Random Forests. In Ref. [214] Breiman proposes a variant of bagging which he calls a *random forest*. Random forest is a general class of ensemble building methods using a decision tree as the base classifier. To be labeled a "random forest" an ensemble of decision trees should be built by generating independent identically distributed random vectors Θ_k, $k = 1, \ldots, L$, and use each vector to grow a decision tree. The formal definition in Ref. [214] is

Definition. A random forest is a classifier consisting of a collection of tree-structured classifiers, each tree grown with respect to a random vector Θ_k, where Θ_k, $k = 1, \ldots, L$, are independent and identically distributed. Each tree casts a unit vote for the most popular class at input $\mathbf{x}$.

Thus a random forest could be built by sampling from the feature set, from the data set, or just varying randomly some of the parameters of the tree.[26] Any combination of these sources of diversity will also lead to a random forest. For example, we may sample from the feature set *and* from the data set as well [21]. In this way the random vector Θ_k will be a concatenation of two bootstrap samples, one from the training set of data points and one from the set of features, containing $N + n$ elements altogether.

One of the most successful heuristics within the random forest family is the random input selection. Along with selecting bootstrap samples from the training data, random feature selection is carried out *at each node of the tree*. We choose randomly a subset S with M features from the original set of n features and seek within S the best feature to split the node. A feature subset is selected anew for each node.

[26] A feature-selection random forest is discussed next in Chapter 8.

Breiman suggests to grow in this way a full CART tree (no pruning). The recommended value of M is $\lfloor \log_2 n + 1 \rfloor$ [214], where n is the total number of features.

7.1.3.2 Pasting Small Votes. Massive data sets are more common now than they were in the early days of pattern recognition in the 1960s and 1970s. Availability of excessive and continuously growing computing resources allowed huge amounts of data to be stored, for example, in retail, e-commerce, financial markets, bio-informatics, industry, communications, and so on. The focus is being shifted from "how to re-use the available data set so that the conclusions be valid?" towards "how to handle the data set if it does not fit into the computer's memory?"

Breiman [215] suggests using the so-called *small votes* whereby the individual classifiers are trained on relatively small subsets of the training set called "*bites*" [216]. The training sets are sampled from the large data set either randomly, called *Rvotes* (similar to bagging), or based on importance, called *Ivotes* (similar to boosting which we will see later in this chapter). Pasting small votes is suitable for on-line learning as the ensemble can be updated by adding a new classifier each time a sufficient amount of new incoming training data has been accumulated.

Let $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ be the available labeled data set (suppose N is about one hundred thousand), and M be the size of the "bite" (for example, 200).

One possible scenario for building the ensemble $\mathcal{D}$ is to fix in advance the number of classifiers, L. Take L random samples of size M from $\mathbf{Z}$, with replacement (Rvotes), and build a classifier on each sample. Use majority vote to infer the class label. We can use a validation set to estimate the accuracy of $\mathcal{D}$ and optimize the ensemble with respect to L and M. Pasting Rvotes is simple but has been found disappointingly inaccurate compared to pasting Ivotes [215,216].

Pasting Ivotes requires an estimate of the generalization accuracy at each building step. The new training set is sampled from $\mathbf{Z}$ (with replacement) according to the current accuracy. The idea is that approximately half of the elements of the new "bite" should be "easy" and half should be "difficult." Suppose that l classifiers have been built, so the current ensemble is $\mathcal{D}_l = \{D_1, \ldots, D_l\}$ and the estimate of the generalization error of $\mathcal{D}_l$ is e_l (assume $e_l < 0.5$). The training set for the new classifier D_{l+1} is sampled from $\mathbf{Z}$ according to the following procedure.

1. Take randomly an element $\mathbf{z}_j$ of $\mathbf{Z}$ (use uniform distribution).
2. Identify the classifiers in $\mathcal{D}_l$ that did not have $\mathbf{z}_j$ in their training sets, called *out-of-bag* classifiers. If $\mathbf{z}_j$ was in all training sets, then ignore it and pick another element of $\mathbf{Z}$.
3. Run $\mathbf{z}_j$ through the out-of-bag classifiers and take the majority vote between them to find a guessed label for $\mathbf{z}_j$.
4. If the guessed label is incorrect, then add $\mathbf{z}_j$ to the training set for classifier D_{l+1}. Otherwise, add it to the training set with probability

$$\frac{e_l}{1 - e_l} \tag{7.3}$$

5. Repeat steps (1) to (4) until M elements from $\mathbf{Z}$ are selected as the training set for classifier D_{l+1}.

A similar cascade procedure for filtering the training sets for an ensemble of three neural networks is proposed by Drucker et al. [217].

The next step is to train classifier D_{l+1} on the selected training set. The estimate e_{l+1} needed for selecting the next training set can be calculated in different ways.

Breiman suggests to use the *out-of-bag estimate* whereby e_l is estimated during the selection of the training set for classifier D_{l+1}.[27] Denote by $\tilde{e}$ the ratio of misclassifications to the total number of *tested* elements $\mathbf{z}_j$ (i.e., the nonignored ones). The error e_l is calculated as a smoothed estimate

$$e_l = \alpha e_{l-1} + (1 - \alpha)\tilde{e} \tag{7.4}$$

where $\alpha \in (0, 1)$ is a parameter (recommended value 0.75 [215]).

To construct a bite of size M containing approximately half "easy" and half "difficult" objects selected through the above procedure, we must test on average about $M/2e_{l-1}$ randomly selected objects. If e_{l-1} is large, then the number of tested objects will not be too large, therefore the estimate $\tilde{e}$ might have high variance.

To start the algorithm we first build a classifier on a randomly sampled training set of size M and evaluate e_0 on an out-of-bag testing set. Using e_0 we select a training set for D_1 and simultaneously calculate e_1. With e_1, we select a training set for D_2 and so on.

It has been found in Ref. [215] that e_l is a noisy estimate and sometimes has a systematic bias either above or below the true generalization error for $\mathcal{D}_l$. However, it was also noticed that e_l tracks the generalization error well, so if we stop the training when e_l levels off or starts increasing, the corresponding generalization error should be close to the best possible value too. In the experiments carried out by Breiman N had values from 2000 to 43,000 and M was varied from 100 to 800. Unpruned decision trees were used as the base classifiers.

An alternative, which avoids out-of-bag estimates altogether is to use *a separate validation set* to evaluate e_l. Taking into account that the proposed method is aimed at massive data sets, leaving aside a validation set of a chosen size without diminishing substantially the size of the training set should not be a problem. All the objects in this validation set will be out-of-bag throughout building the ensemble. This will avoid the variability and the systematic bias of the out-of-bag estimate. The smoothing of Eq. (7.4) becomes unnecessary too.

Chawla et al. [216] suggest using a distributed version of pasting small votes. The data set $\mathbf{Z}$ is partitioned into disjoint subsets, and an ensemble of Ivotes is grown *on each subset*. Then $\mathcal{D}$ is formed by pooling all classifiers designed throughout the training. The rationale for this distributed version is mainly a substantial gain in

[27] Thus e_{l-1} must be used at step (4) because e_l is being currently evaluated.

computation time compared to Ivote on the whole of $\mathbf{Z}$, which is itself advantageous compared to AdaBoost on $\mathbf{Z}$. The individual Ivote procedures can be run on separate processors, which do not need to communicate during training. The accuracy of the final ensemble has been found in Ref. [216] to be comparable to that of Ivote or AdaBoost on the whole of $\mathbf{Z}$. This result is not unexpected as using different partitions induces additional diversity in the final ensemble.

Finally, to exploit the computational advantage of distributed Ivotes to the full, we need the appropriate distributed computing resource, for example, a computer cluster.

Example: Pasting Small Votes. To examine the effect of the out-of-bag estimate, we generated a banana data set of $N = 10{,}000$ objects and set the bite size at $M = 150$. Separate validation and testing sets were generated, each containing 200 objects. Effectively, the training set size is 10,200 because the validation set belongs there too. Ensembles of 50 classifiers were evolved. Two experiments were carried out. In the first experiment the error e_l was calculated using the validation set. This estimate was used for selecting the training sets in step (4). The ensemble error was estimated also through the out-of-bag method in order to see whether its shape differs from that of e_l. Figure 7.3 plots the validation e_l, the out-of-bag error e_l and the testing error. In the second experiment, the training sets were generated according to the out-of-bag approximation of e_l and the vali-

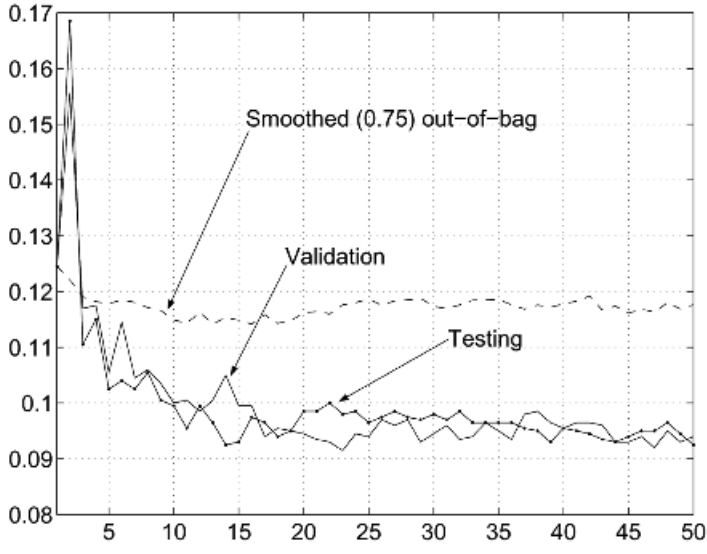


Fig. 7.3 *Error rates for the banana data for the Ivotes algorithm. The estimate of the ensemble error, e_l, needed for filtering the consecutive training sets has been calculated on a validation set.*

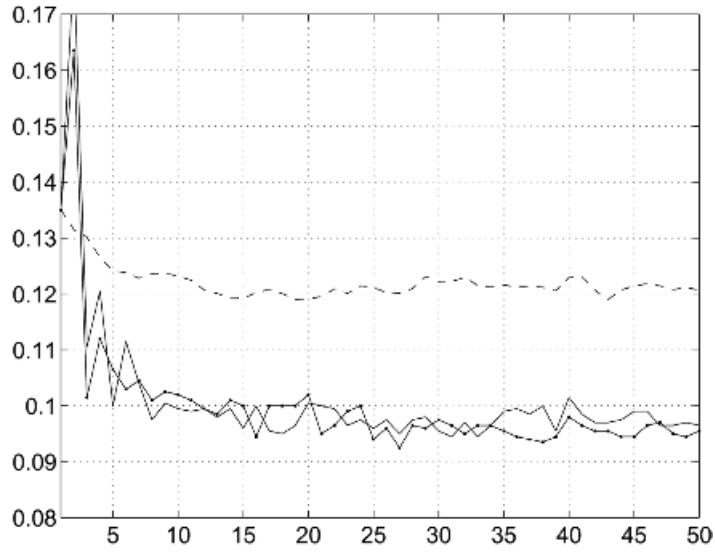


Fig. 7.4 *Error rates for the banana data for the Ivotes algorithm. The estimate of the ensemble error, e_l, needed for filtering the consecutive training sets has been calculated by the smoothed out-of-bag method as in Eq.* (*7.4*).

dation set estimate was calculated for completeness. The results are plotted in Figure 7.4.

Each graph is the average of 10 runs. Independent training/validation/testing data sets were generated for each run from the distribution of the banana problem. The similarity of the testing error curves suggests that the two estimates of e_l lead to practically equivalent ensembles regardless of the differences in the estimated values. As expected, the out-of-bag error was not a very close predictor of the testing error but it picked up its trend and managed to bring it down to the same level as the validation set method.

Skurichina [218] proposes a variant of bagging which she calls "*nice*" bagging. Instead of taking all L classifiers, we only accept classifiers whose training error is smaller than the error made by an individual classifier built on the whole data set. In fact, we may again consider an ensemble of L classifiers by dismissing in the training process the classifiers whose training error is above the threshold and continuing until the ensemble size reaches L. The linear discriminant classifier (LDC) is stable, hence not the best candidate for bagging. However, for data sets where the number of cases is small and the number of features is large, LDC is no longer stable because small changes in the training set might lead to large changes of the classifier. Bagging and "nice bagging" have been found to work for unstable LDC [218].

7.2 BOOSTING

7.2.1 Origins: Algorithm Hedge(β)

Boosting was inspired by an on-line learning algorithm called *Hedge*(β) [219]. This algorithm allocates weights to a set of strategies used to predict the outcome of a certain event. The weight of strategy s_i, if properly scaled, can be interpreted as the probability that s_i is the best (most accurate) predicting strategy in the group. The distribution is updated on-line after each new outcome. Strategies with the correct prediction receive more weight while the weights of the strategies with incorrect predictions are reduced.

At this point we shall relate Hedge(β) to our classifier combination set-up, although somewhat differently to the correspondence suggested in Ref. [219]. Here the *strategies* will correspond to the classifiers in the ensemble and the *event* will correspond to labeling of a randomly drawn $\mathbf{z}_j$ from $\mathbf{Z}$. Suppose that a classifier ensemble $\mathcal{D} = \{D_1, \ldots, D_L\}$ is available but we do not know how well each classifier works on the problem in question. A point is sampled from $\mathbf{Z}$ and each classifier gives a prediction for its label. Whose prediction shall we choose? At first we have no reason to prefer one classifier or another, so we pick at random a classifier from $\mathcal{D}$ and take its decision. Define the *loss* to be 1 if the class label is wrong and 0 if it is correct. The expected loss from our random choice of a classifier will be the average number of classifiers in $\mathcal{D}$ labeling the selected point incorrectly. Since we want to improve the prediction for the next point we draw, it is reasonable to increase the probability of selecting one of the classifiers with the most correct predictions on the points previously seen. Thus we alter the distribution on $\mathcal{D}$ as more examples are classified. If we knew in advance which is the best classifier in the team and always went along with its decision, then the incurred loss (the error across all tested examples) will be the smallest one possible. However, we assume no such knowledge, and therefore apply the Hedge(β) algorithm.

Suppose we make a prediction by randomly selecting a classifier from $\mathcal{D}$ *according to a probability distribution* on $\mathcal{D}$. The algorithm Hedge(β) evolves such a distribution on $\mathcal{D}$ in order to minimize the cumulative loss of the prediction. The distribution is calculated by normalizing a set of weights that are updated after a presentation of $\mathbf{z}_j$ from the data set $\mathbf{Z}$. The weights for the classifiers that misclassify $\mathbf{z}_j$ (incur loss 1) are diminished by a prespecified multiplier β while the other weights will stay the same (loss 0). When the distribution is recalculated through Eq. (7.5), the probabilities for the successful classifiers will increase correspondingly. The larger the number of trials, N (data points in $\mathbf{Z}$), the sharper the peak of p_i^{N+1} will be about the most accurate classifier. The *Hedge*(β) *algorithm* is described in Figure 7.5. Freund and Schapire prove an upper bound on the loss, which is not much worse than the loss of the best classifier in the ensemble [219].

More generally we can assume that the loss l_i^j is a number within the interval [0, 1] rather than just 0 or 1. Consequently, we have continuous-valued $\lambda_i \in$

HEDGE (β)

Given:

- $\mathcal{D} = \{D_1, \ldots, D_L\}$: the classifier ensemble (L strategies)
- $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$: the data set ($N$ trials).

1. Initialize the parameters
   - Pick $\beta \in [0, 1]$.
   - Set the weights $\mathbf{w}^1 = [w_1, \ldots, w_L]$, $w_i^1 \in [0, 1]$, $\sum_{i=1}^{L} w_i^1 = 1$. $\left(\text{Usually } w_i^1 = \frac{1}{L}\right)$.
   - Set $\Lambda = 0$ (the cumulative loss).
   - Set $\lambda_i = 0, i = 1, \ldots, L$ (the individual losses).
2. For every $\mathbf{z}_j, j = 1, \ldots, N$,
   - Calculate the distribution by

$$p_i^j = \frac{w_i^j}{\sum_{k=1}^{L} w_k^j}, \quad i = 1, \ldots, L. \tag{7.5}$$

   - Find the L individual losses. ($l_i^j = 1$ if D_i misclassifies $\mathbf{z}_j$ and $l_i^j = 0$ if D_i classifies $\mathbf{z}_j$ correctly, $i = 1, \ldots, L$).
   - Update the cumulative loss

$$\Lambda \leftarrow \Lambda + \sum_{i=1}^{L} p_i^j \, l_i^j \tag{7.6}$$

   - Update the individual losses

$$\lambda_i \leftarrow \lambda_i + l_i^j. \tag{7.7}$$

   - Update the weights

$$w_i^{j+1} = w_i^j \, \beta^{l_i^j}. \tag{7.8}$$

3. Calculate the return Λ, λ_i, and p_i^{N+1}, $i = 1, \ldots, L$.

Fig. 7.5 *Algorithm Hedge(β).*

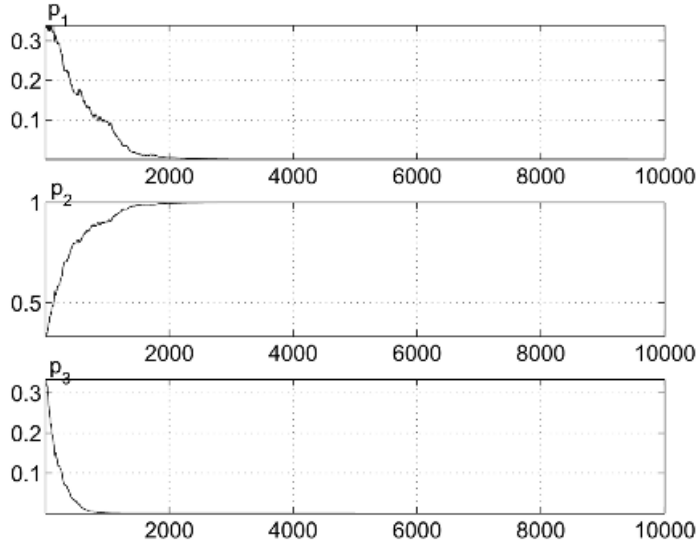


Fig. 7.6 *Change of the distribution on the set of three classifiers (LDC, Parzen, and a decision tree) on the banana data by the Hedge(β) algorithm.*

$[0, N]$ and $\Lambda \in [0, L \times N]$. Freund and Schapire show that if we pick

$$\beta = g(\tilde{\lambda}/\ln L), \qquad \text{where} \quad g(\zeta) = \frac{1}{1+\sqrt{\frac{2}{\zeta}}} \tag{7.9}$$

and $\tilde{\lambda}$ is a guess for an upper bound of the loss of the best classifier, then the total loss is bounded from above as follows

$$\Lambda \leq \min_{i=1}^{L} \lambda_i + \sqrt{2\tilde{\lambda} \ln L} + \ln L \tag{7.10}$$

Example: Bounds on the Loss for Hedge(β). An ensemble of three classifiers is considered for the banana data: a linear discriminant classifier (LDC), a Parzen classifier, and a prepruned decision tree. One hundred data points were generated for training and 10,000 more points for testing. The classifiers trained on the 100 training points gave the following testing error: LDC 19.00 percent, Parzen 10.09 percent, and the tree classifier 19.82 percent.[28] Then we "guessed" correctly the minimum value of the error, $\tilde{\lambda} = 10{,}000 \times 0.1009 = 1009$, and substituted it into Eq. (7.10). The upper bound obtained for the total loss Λ was 1088. Using $\tilde{\lambda}$ we also calculated $\beta = 0.9554$ using Eq. (7.9). Applying the Hedge(β) algorithm on the testing data, the observed loss Λ was 1034 ($1009 \leq 1034 \leq 1088$). Figure 7.6

[28] A CART tree classifier was constructed (see Chapter 2) using a prepruning option (early stopping) with a threshold value of 4.

shows how the probabilities p_1, p_2, and p_3 changed along with the new examples (data points from the testing set) being presented.

The probability corresponding to the least accurate classifier, D_3, dropped to zero first, next was the probability for D_1, and after the 2000th object the ensemble practically consisted of the best classifier D_2. The further we run the experiment (larger N), the closer the total loss will be to that of the best classifier in the ensemble.

7.2.2 AdaBoost Algorithm

Boosting is defined in Ref. [219] as related to the "general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb." The general boosting idea is to develop the classifier team $\mathcal{D}$ incrementally, adding one classifier at a time. The classifier that joins the ensemble at step k is trained on a data set selectively sampled from the training data set $\mathbf{Z}$. The sampling distribution starts from uniform, and progresses towards increasing the likelihood of "difficult" data points. Thus the distribution is updated at each step, increasing the likelihood of the objects misclassified at step $k-1$. Here the correspondence with Hedge(β) is transposed. The classifiers in $\mathcal{D}$ are the trials or events, and the data points in $\mathbf{Z}$ are the strategies whose probability distribution we update at each step. The algorithm is called *AdaBoost* in Ref. [219] which comes from ADAptive BOOSTing. There are two implementations of AdaBoost: with *reweighting* and with *resampling*. The description above refers to the resampling implementation. For the reweighting implementation we assume that the base classifiers can directly use the probabilities on $\mathbf{Z}$ as weights. No sampling is needed in this case, so the algorithm becomes completely *deterministic*.

AdaBoost was proposed initially for two classes and then extended for multiple classes. Figure 7.7 shows *AdaBoost.M1*, which is the most straightforward multiclass extension of AdaBoost [219]. The resampling implementation is shown.

Example: Illustration of AdaBoost. The performance of the AdaBoost algorithm is illustrated on the *forensic glass data* from UCI. The experimental set-up was chosen as in Ref. [214]: 100 runs with a split of the data into 90 percent training and 10 percent testing. Fifty CART trees were constructed as the ensemble members, each tree grown in full (no pruning). The testing accuracy averaged across the 100 runs is shown in Figure 7.8. The graph demonstrates the remarkable ability of AdaBoost to reduce the error without overfitting in spite of the progressively increasing complexity of the ensemble.

7.2.3 arc-x4 Algorithm

Breiman studies bagging and boosting from various curious angles in Ref. [116]. He calls the class of boosting algorithms *arcing* algorithms as an acronym for "adaptive resample and combining." His *arc-fs* algorithm is AdaBoost (named "fs" after its authors Freund and Schapire [219]). Breiman proposes a boosting algorithm called *arc-x4* to investigate whether the success of AdaBoost roots in its technical details or

ADABOOST.M1

Training phase

1. Initialize the parameters
   - Set the weights $\mathbf{w}^1 = [w_1, \ldots, w_N]$, $w_j^1 \in [0, 1]$, $\sum_{j=1}^{N} w_j^1 = 1$. (Usually $w_j^1 = \frac{1}{N}$).
   - Initialize the ensemble $\mathcal{D} = \emptyset$.
   - Pick L, the number of classifiers to train.
2. For $k = 1, \ldots, L$
   - Take a sample S_k from $\mathbf{Z}$ using distribution $\mathbf{w}^k$.
   - Build a classifier D_k using S_k as the training set.
   - Calculate the weighted ensemble error at step k by

$$\epsilon_k = \sum_{j=1}^{N} w_j^k l_k^j, \tag{7.11}$$

   ($l_k^j = 1$ if D_k misclassifies $\mathbf{z}_j$ and $l_k^j = 0$ otherwise.)
   - If $\epsilon_k = 0$ or $\epsilon_k \geq 0.5$, ignore D_k, reinitialize the weights w_j^k to $\frac{1}{N}$ and continue.
   - Else, calculate

$$\beta_k = \frac{\epsilon_k}{1 - \epsilon_k}, \quad \text{where} \quad \epsilon_k \in (0, 0.5), \tag{7.12}$$

   - Update the individual weights

$$w_j^{k+1} = \frac{w_j^k \beta_k^{(1-l_k^j)}}{\sum_{i=1}^{N} w_i^k \beta_k^{(1-l_k^i)}}, \quad j = 1, \ldots, N. \tag{7.13}$$

3. Return $\mathcal{D}$ and $\beta_1, \ldots, \beta_L$.

Classification phase

4. Calculate the support for class ω_t by

$$\mu_t(\mathbf{x}) = \sum_{D_k(\mathbf{x})=\omega_t} \ln\left(\frac{1}{\beta_k}\right). \tag{7.14}$$

5. The class with the maximum support is chosen as the label for $\mathbf{x}$.

Fig. 7.7 *The AdaBoost.M1 algorithm with resampling.*

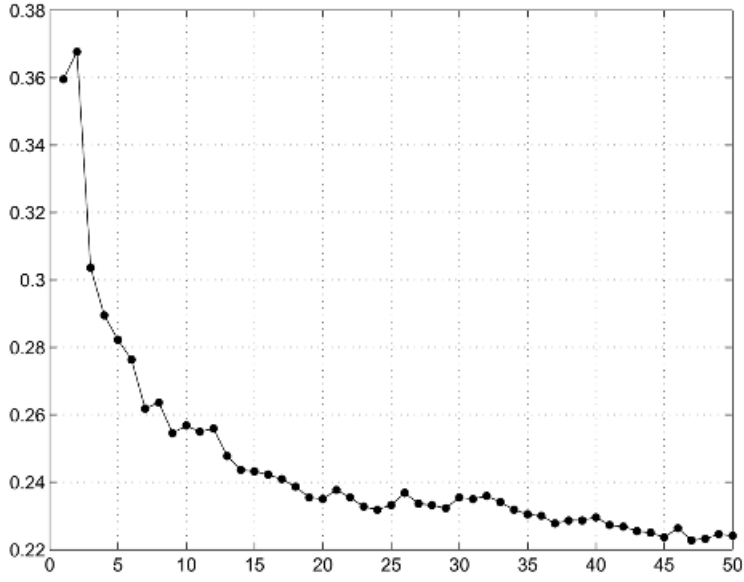


Fig. 7.8 *Testing error of AdaBoost versus the ensemble size for the forensic glass data set.*

in the resampling scheme it uses. The difference between AdaBoost and arc-x4 is twofold. First, the weight for object $\mathbf{z}_j$ at step k is calculated as the proportion of times $\mathbf{z}_j$ has been misclassified by the $k-1$ classifiers built so far. Second, the final decision is made by plurality voting rather than weighted majority voting. The arc-x4 algorithm is described in Figure 7.9.

Breiman admits that while AdaBoost originates from a theory, arc-x4 is an ad hoc algorithm [116]. The parameter of the algorithm, the power of the number m_j, has been fixed to the constant 4 (hence the name) by a small experiment. Yet arc-x4 has been found to be as good as AdaBoost. Breiman compares the behaviors of AdaBoost and arc-x4 and finds that AdaBoost makes more abrupt moves while arc-x4 has a more gradual behavior. This is reflected, for example, by the standard deviations of the weights assigned to a single data point. This standard deviation was found to be much larger for AdaBoost than for arc-x4.

7.2.4 Why Does AdaBoost Work?

One of the explanations of the success of AdaBoost comes from the algorithm's property to drive the ensemble training error to zero very quickly, practically in the first few iterations.

7.2.4.1 The Upper Bound on the Training Error. Freund and Schapire prove an upper bound on the training error of AdaBoost [219] for the case of two classes. The following theorem gives their result

ARC-X4

Training phase

1. Initialize the parameters
   - Set the weights $\mathbf{w}^1 = [w_1, \ldots, w_N]$, $w_j^1 \in [0, 1]$, $\sum_{j=1}^{N} w_j^1 = 1$. (Usually $w_j^1 = \frac{1}{N}$).
   - Initialize the ensemble $\mathcal{D} = \emptyset$.
   - Pick L, the number of classifiers to train.
2. For $k = 1, \ldots, L$
   - Take a sample S_k from $\mathbf{Z}$ using distribution $\mathbf{w}^k$.
   - Build a classifier D_k using S_k as the training set.
   - Find m_j as the proportion of classifiers currently in the ensemble which misclassify $\mathbf{z}_j$. Update the individual weights

$$w_j^{k+1} = \frac{1 + m_j^4}{\sum_{i=1}^{N} 1 + m_i^4}, \quad j = 1, \ldots, N. \tag{7.15}$$

3. Return $\mathcal{D}$.

Classification phase

4. Run $D_1, \ldots, D_L$ on the input $\mathbf{x}$.
5. The class with the maximum number of votes is chosen as the label for $\mathbf{x}$.

Fig. 7.9 *The arc-x4 algorithm.*

Theorem 7.1. Let $\Omega = \{\omega_1, \omega_2\}$. Let ε be the ensemble training error and let ε_i, $i = 1, \ldots, L$, be the weighted training errors of the classifiers in $\mathcal{D}$ as in Eq. (7.11). Then

$$\varepsilon < 2^L \prod_{i=1}^{L} \sqrt{\varepsilon_i (1 - \varepsilon_i)} \tag{7.16}$$

The proof is reproduced in Appendix 7A with small changes so that it can be used directly for multiple classes. The result of Theorem 7.1 indicates that by adding more classifiers the training error of the ensemble approaches zero.

The error bound for the multiple class case is the same as for the two-class case as long as the classifiers have individual errors smaller than $\frac{1}{2}$. AdaBoost.M1 takes this

into account by the clause in step 2, which reinitializes the weights in case $\varepsilon_i \geq 0.5$. The following Theorem 7.2 holds for the general case of c classes.

Theorem 7.2. Let $\Omega = \{\omega_1, \ldots, \omega_c\}$. Let ε be the ensemble training error and let ε_i, $i = 1, \ldots, L$ be the weighted training errors of the classifiers in $\mathcal{D}$ as in Eq. (7.11) and $\varepsilon_i < \frac{1}{2}$. Then

$$\varepsilon < 2^L \prod_{i=1}^{L} \sqrt{\varepsilon_i(1-\varepsilon_i)} \tag{7.17}$$

The proof is given in Appendix 7B.

Freund and Schapire argue that having an error greater than half is too strict a demand for a multiple-class *weak learner*. They proceed to propose another version of AdaBoost, called AdaBoost.M2, which does not require $\varepsilon_i < 0.5$ [219]. Note that ε_i is *not* the error of classifier D_i. It is a *weighted* error. This means that if we applied D_i on a data set drawn from the problem in question, its (conventional) error could be different from ε_i; it could be larger or smaller.

7.2.4.2 The Margin Theory. Experiments with AdaBoost showed an unexpected phenomenon: the testing error continues to decrease with adding more classifiers even after the training error reaches zero. This prompted another look into the possible explanations and brought forward the margin theory [213,220].

The concept of *margins* comes from statistical learning theory [221] in relation to the Vapnik–Chervonenkis dimension (VC-dimension). In layman terms, the VC-dimension gives an upper bound on the classification ability of classifier models. Although the bound is loose, it has proven to be an important theoretical accessory in pattern recognition and machine learning. The *support vector machine* (SVM) classifier is underpinned by the statistical learning theory and in particular by the idea of maximizing the margins. Intuitively, the margin for an object is related to the certainty of its classification. Objects for which the assigned label is correct and highly certain will have large margins. Negative margins signify incorrect classification. Objects with uncertain classification are likely to have small margins. A small margin will cause instability of the classification label, that is, the object might be assigned to different classes by two similar classifiers.

For c classes, the margin of object $\mathbf{x}$ is calculated using the degree of support $\mu_j(\mathbf{x})$, $j = 1, \ldots, c$, as

$$m(\mathbf{x}) = \mu_k(\mathbf{x}) - \max_{j \neq k} \{\mu_j(\mathbf{x})\} \tag{7.18}$$

where ω_k is the (known) class label of $\mathbf{x}$ and $\sum_{j=1}^{c} \mu_j(\mathbf{x}) = 1$.

Thus all objects that are misclassified will have negative margins, and those correctly classified will have positive margins. Trying to maximize the margins (called

"boosting the margins") will intuitively lead to "more confident" classifiers. Schapire et al. [213] prove upper bounds on the testing error that depend on the margin. The main theoretical result for $c = 2$ classes is given in the following theorem.

Theorem 7.3. Let $\mathcal{H}$ be a finite space of base classifiers.[29] For any $\delta > 0$ and $\theta > 0$, with probability at least $1 - \delta$ over the random choice of the training set $\mathbf{Z}$, any classifier ensemble $\mathcal{D} = \{D_1, \ldots, D_L\} \subseteq \mathcal{H}$ combined by the weighted average satisfies

$$P(\text{error}) \leq P(\text{training margin} \leq \theta) \tag{7.19}$$

$$+ O\left(\frac{1}{\sqrt{N}}\left(\frac{\log N \log |\mathcal{H}|}{\theta^2} + \log(1/\delta)\right)^{1/2}\right) \tag{7.20}$$

where $P(\text{error})$ is the probability that the ensemble will make an error in labeling $\mathbf{x} \in \Re^n$, drawn randomly from the distribution of the problem and $P(\text{training margin} \leq \theta)$ is the probability that the margin for a randomly drawn data point from a randomly drawn training set does not exceed θ. $|\mathcal{H}|$ is the cardinality of $\mathcal{H}$.

For the more general case of finite or infinite $\mathcal{H}$ with VC dimension d, the following bound holds, assuming that $N \geq d \geq 1$

$$P(\text{error}) \leq P(\text{training margin} \leq \theta) \tag{7.21}$$

$$+ O\left(\frac{1}{\sqrt{N}}\left(\frac{d \log^2(N/d)}{\theta^2} + \log(1/\delta)\right)^{1/2}\right) \tag{7.22}$$

Neither bound depends on the number of classifiers in the ensemble. Although the bounds are quite loose, they show the tendency that larger margins lead to a smaller upper bound on the testing error.

Example: Classification Margins. To illustrate the effect of bagging and AdaBoost on the margins, we ran both algorithms for the rotated check-board data (Figure 1.10) as in the earlier example. The margins were calculated according to Eq. (7.18). For visualization purposes Schapire et al. propose *margin distribution graphs* showing the cumulative distribution of the margins for a given data set. The x-axis is the margin, m, and the y-axis is the number of points whose margin is less than or equal to m. If all training points are classified correctly, there will be only positive margins. Ideally, all points should be classified correctly so that

[29] A finite space of base classifiers is, for example, the set of all decision trees of a given size over a set of discrete features. For example, the set of all decision stumps over a set of n binary features contains n elements (classifiers), one for each feature.

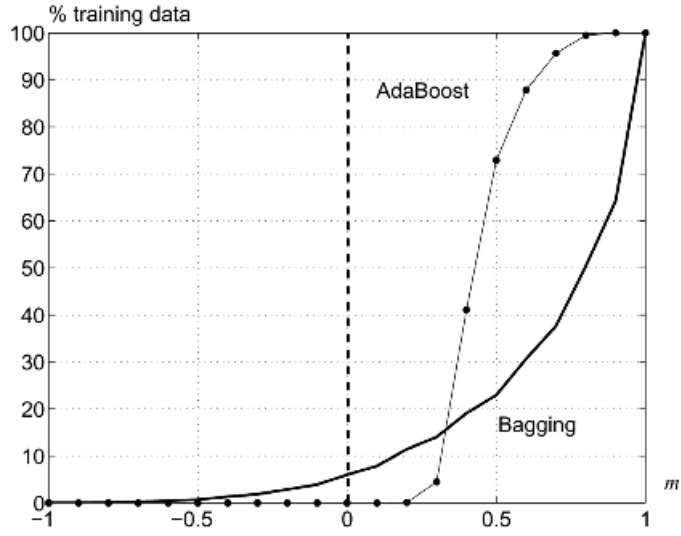


Fig. 7.10 *Margin distribution graphs for bagging and AdaBoost for the rotated check-board example.*

all margins are positive. If all points are classified correctly and with the maximum possible certainty, the cumulative graph will be a single vertical line at $m = 1$. Figure 7.10 shows the margin distribution graphs for bagging and AdaBoost.[30] The training and testing error rates for bagging and AdaBoost are shown in Figure 7.11. The dominance of AdaBoost over bagging shows that larger margins are associated with lower testing error.

7.2.5 Variants of Boosting

There is a great variety of methods drawn upon the basic idea of boosting. Owing to its success, boosting is probably the most rapidly growing subarea of classifier combination. Versions of AdaBoost for real-valued classifier outputs, multiclass problems, and ECOC ensembles[31] are proposed [219,222,223]. Explanations for the remarkably good performance of boosting have been sought in its relationship with logistic regression [224–226]. This relationship gave rise to variants such as *LogitBoost* and a parametarized family of iterative algorithms developed in Ref. [224]. A procedure called *DOOM* (direct optimization of margins) is proposed Ref. [227]. There are also many ad hoc variants supported by empirical evidence including MultiBoosting [228], AveBoost [229], AdaBoost-VC [230] and arc-x4 [116].

[30] In bagging, the support for class ω_j, $\mu_j(\mathbf{x})$, was calculated as the proportion of votes for that class.
[31] ECOC stands for "error correcting output code." This concept is detailed in Chapter 8.

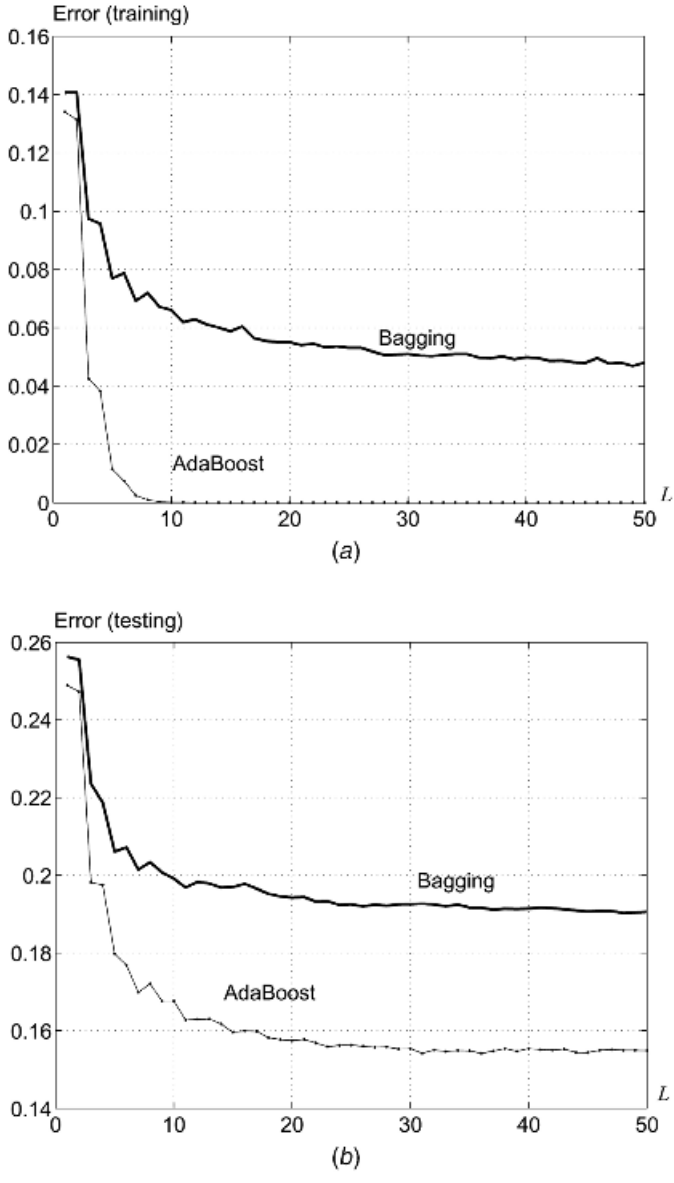


Fig. 7.11 *Training and testing error of bagging and AdaBoost for the rotated check-board example.*

7.3 BIAS-VARIANCE DECOMPOSITION

One of the explanations for the success of bagging and boosting roots is the *bias-variance* decomposition of the classification error.

7.3.1 Bias, Variance, and Noise of the Classification Error

Let D be a classifier randomly chosen from a population of classifiers. Consider a single point $\mathbf{x}$ in the feature space $\Re^n$. The *state of the nature* at $\mathbf{x}$ is a random variable $\omega \in \Omega$ (the class label of $\mathbf{x}$). The classifier tries to guess the state of the nature by picking one label from Ω. Suppose that the true values of the posterior probabilities for the c classes, given $\mathbf{x}$, are $P(\omega_i|\mathbf{x})$, and the probabilities across all possible D are $P_D(\omega_i|\mathbf{x})$ (the guessed state of the nature), $i = 1, \ldots, c$. $P_D(\omega_i|\mathbf{x})$ is the probability that a randomly chosen classifier D will assign ω_i for the given $\mathbf{x}$. By definition $\sum_i P(\omega_i|\mathbf{x}) = 1$ and $\sum_i P_D(\omega_i|\mathbf{x}) = 1$.

7.3.1.1 Kohavi–Wolpert Definitions. Kohavi and Wolpert [231] suggest the following definitions for bias, variance, and noise at a given $\mathbf{x}$

$$\text{bias} = \frac{1}{2}\sum_{\omega_i} (P(\omega_i|\mathbf{x}) - P_D(\omega_i|\mathbf{x}))^2 \tag{7.23}$$

$$\text{variance} = \frac{1}{2}\left(\sum_{\omega_i} 1 - P_D(\omega_i|\mathbf{x})^2\right) \tag{7.24}$$

$$\text{noise} = \frac{1}{2}\left(\sum_{\omega_i} 1 - P(\omega_i|\mathbf{x})^2\right) \tag{7.25}$$

The general concept of *bias* of an estimate is the averaged difference between the true and the predicted values. In our case the bias can be regarded as some measure of the difference between the true distribution $P(\omega_i|\mathbf{x})$ and the guessed distribution $P_D(\omega_i|\mathbf{x})$, $i = 1, \ldots, c$.

The *variance* term expresses the variability of the classifier's guess regardless of the true state of the nature. A conventional measure of variability of a random variable is its variance. However, there is no clear definition of variance for nominal variables such as the class label. To measure variability we can use the entropy of the distribution

$$H = -\sum_{\omega_i} P_D(\omega_i|\mathbf{x}) \log P_D(\omega_i|\mathbf{x}) \tag{7.26}$$

$H = 0$ will signify no variability whereas $H = \log c$ will correspond to the highest variability where each label has the same probability of $1/c$. Kohavi and Wolpert use the Gini index (2.65) introduced in Chapter 2 as the variance; that is,

$$G = \sum_{\omega_i} 1 - P_D(\omega_i|\mathbf{x})^2 \tag{7.27}$$

The term *noise* measures the variability of the state of the nature regardless of the guess of the classifier. Again, we can use for that H or G.

The scaling constant of $\frac{1}{2}$ is needed so that the sum of the three components gives exactly the classifier error as discussed later.

7.3.1.2 Breiman's Definitions. Breiman [232] gives a different definition of variance, noise, and bias. Suppose ω^* is the class label with the highest true probability (given $\mathbf{x}$). The noise term is equivalent to the Bayes error for that $\mathbf{x}$; that is,

$$\text{noise} \equiv 1 - P(\omega^*|\mathbf{x}) \tag{7.28}$$

This term gives the error for $\mathbf{x}$ if the most probable class ω^* is always chosen as its label. Note the difference with the Kohavi–Wolpert noise, which is the error for $\mathbf{x}$ in case we pick a random class label from the true distribution $P(\omega_i|\mathbf{x})$.

Let $\hat{\omega^*}$ be the class label with the largest $P_D(\omega_i|\mathbf{x})$, that is, the most likely output for $\mathbf{x}$ of a random classifier D. The bias of D is defined as the additional amount of error incurred when $\hat{\omega^*}$ is assigned to $\mathbf{x}$ weighted by the probability of this class assignment, $P_D(\hat{\omega^*}|\mathbf{x})$

$$\text{bias} \equiv (P(\omega^*|\mathbf{x}) - P(\hat{\omega^*}|\mathbf{x}))P_D(\hat{\omega^*}|\mathbf{x}) \tag{7.29}$$

Note that the bias is always nonnegative because ω^* maximizes $P(\omega_i|\mathbf{x})$.

Breiman calls the term corresponding to variance "*spread*." The spread shows how much the guessed distribution varies across class labels other than $\hat{\omega^*}$ and ω^*

$$\text{spread} \equiv \sum_{\omega_i \neq \hat{\omega^*}} (P(\omega^*|\mathbf{x}) - P(\omega_i|\mathbf{x}))P_D(\omega_i|\mathbf{x}) \tag{7.30}$$

7.3.1.3 Domingos' Definitions. Domingos [233–235] proposes a unified definition of bias, variance, and noise, which encompasses both previously introduced definitions. Let $l(T(\mathbf{x}), D(\mathbf{x}))$ be the loss for $\mathbf{x}$ incurred by a randomly picked classifier D where $T(\mathbf{x})$ is the true label of $\mathbf{x}$ and $D(\mathbf{x})$ is the guessed label. Domingos defines the bias for $\mathbf{x}$ to be

$$\text{bias} \equiv l(\omega^*, \hat{\omega^*}) \tag{7.31}$$

Thus the bias does not depend on the particular realization of the (random) classifier D. The bias only depends on the most probable guessed label (majority label) $\hat{\omega^*}$ and the optimal class label for $\mathbf{x}$, ω^*. For a $0/1$ loss function the bias is either 0 (matched labels) or 1 (mismatched labels). The variance is defined as

$$\text{variance} \equiv \mathcal{E}_D\Big(l(\hat{\omega^*}, D(\mathbf{x}))\Big) \tag{7.32}$$

where $\mathcal{E}_D$ stands for the expectation on all possible realizations of D. The variance is thus a measure of the variation of the guessed label about the majority prediction. If

the same label is always assigned to $\mathbf{x}$, then the loss $l(\hat{\omega^*}, D(\mathbf{x}))$ will be zero because no classifier deviates from the majority prediction (assuming $0/1$ loss). Alternatively, if the most probable guessed label is completely random, that is, with probability $1/c$, the loss will take its maximum value of $1 - (1/c) = (c-1)/c$. The formula for the variance for $0/1$ loss is

$$\text{variance} \equiv \sum_{\omega_i \neq \hat{\omega^*}} P_D(\omega_i|\mathbf{x}) = 1 - P_D(\hat{\omega^*}|\mathbf{x}) \tag{7.33}$$

The noise is defined to be

$$\text{noise} \equiv \mathcal{E}_T(l(T(\mathbf{x}), \omega^*)) \tag{7.34}$$

Thus the noise is only dependent on the problem in question and not on any classifier. For $0/1$ loss,

$$\text{noise} \equiv 1 - P(\omega^*|\mathbf{x}) \tag{7.35}$$

We can think of classifier D as a model. The *optimal* model will always output the label with the highest $P(\omega_i|\mathbf{x})$, that is, ω^*. Then the bias measures how far the majority prediction is from the optimal prediction, the variance shows the variability of the predicted label about the majority prediction, and the noise tells us how far the optimal prediction is from the truth (Bayes error).

Example: Illustration of Bias, Variance (Spread), and Noise. Table 7.1 shows two distributions on $\Omega = \{\omega_1, \ldots, \omega_5\}$ for a fixed $\mathbf{x}$. For this example $\omega^* = \omega_5$ and $\hat{\omega^*} = \omega_4$.

Table 7.2 shows the bias, variance/spread, and noise calculated according to the three definitions. For example, Breiman's bias is calculated as

$$\begin{aligned} \text{bias} &= (P(\omega^*|\mathbf{x}) - P(\hat{\omega^*}|\mathbf{x}))P_D(\hat{\omega^*}|\mathbf{x}) \\ &= (0.4 - 0.2) \times 0.6 = 0.12 \end{aligned}$$

TABLE 7.1 True and Guessed Distributions of the Class Labels for the State of the Nature for a Particular (Fixed) x.

	ω_1	ω_2	ω_3	ω_4	ω_5
True	0.3	0.1	0.0	0.2	0.4
Guessed	0.0	0.1	0.1	0.6	0.2

TABLE 7.2 Bias, Variance (Spread), and Noise for x.

	Bias	Variance (spread)	Noise
Kohavi–Wolpert [230]	0.15	0.29	0.35
Breiman [18]	0.12	0.07	0.60
Domingos [234]	1.00	0.40	0.60

Note that the three components for both Kohavi–Wolpert definition and Breiman's definition sum to 0.79, which is the probability of error, given $\mathbf{x}$

$$\begin{aligned} P(\text{error}|\mathbf{x}) &= 1 - \sum_{\omega_i} P(\omega_i|\mathbf{x}) P_D(\omega_i|\mathbf{x}) \\ &= 1 - (0.3 \times 0.0 + 0.1 \times 0.1 + 0.0 \times 0.1 + 0.2 \times 0.6 + 0.4 \times 0.2) \\ &= 0.79. \end{aligned}$$

Domingos' bias, variance, and noise do not sum to the probability of error. The only case where they do is when we use a square loss function, that is, $l(a,b) = (a-b)^2$. This function is not suitable in classification problems because the output variables are nominal, and difference is not straightforward to define in this case.

7.3.2 Decomposition of the Error

The probability of error for a given $\mathbf{x}$ is the probability of disagreement between the decision by a randomly picked classifier D and the true class label of $\mathbf{x}$

$$P(\text{error}|\mathbf{x}) = 1 - \sum_{\omega_i} P(\omega_i|\mathbf{x}) P_D(\omega_i|\mathbf{x}) \tag{7.36}$$

The total error can be then calculated across the whole feature space $\Re^n$ as

$$P(\text{error}) = \int_{\Re^n} P(\text{error}|\mathbf{x}) p(\mathbf{x})\, d\mathbf{x} \tag{7.37}$$

7.3.2.1 Kohavi–Wolpert's Decomposition. Below we develop Eq. (7.36) to show that it is equal to the sum of bias, variance, and noise. To reach the decomposition form we add and subtract the halved sum of squares of the true and the guessed

distributions and rearrange the terms,

$$P(\text{error}|\mathbf{x}) = 1 - \sum_{\omega_i} P(\omega_i|\mathbf{x})P_D(\omega_i|\mathbf{x}) + \frac{1}{2}\sum_{\omega_i} P(\omega_i|\mathbf{x})^2 + \frac{1}{2}\sum_{\omega_i} P_D(\omega_i|\mathbf{x})^2$$

$$- \frac{1}{2}\sum_{\omega_i} P(\omega_i|\mathbf{x})^2 - \frac{1}{2}\sum_{\omega_i} P_D(\omega_i|\mathbf{x})^2 \tag{7.38}$$

$$= \frac{1}{2}\left(\sum_{\omega_i} (P(\omega_i|\mathbf{x}) - P(\omega_i|\mathbf{x}))^2\right) + \frac{1}{2}\left(1 - \sum_{\omega_i} P_D(\omega_i|\mathbf{x})^2\right)$$

$$+ \frac{1}{2}\left(1 - \sum_{\omega_i} P(\omega_i|\mathbf{x})^2\right) \tag{7.39}$$

$$= \text{bias} + \text{variance} + \text{noise} \tag{7.40}$$

7.3.2.2 Breiman's Decomposition. We start again from the conditional probability of error and add and subtract $P(\omega^*|\mathbf{x})$. Then we use $\sum_i P_D(\cdot) = 1$ as a multiplier to $P(\omega^*|\mathbf{x})$.

$$P(\text{error}|\mathbf{x}) = 1 - \sum_{\omega_i} P(\omega_i|\mathbf{x})P_D(\omega_i|\mathbf{x}) \tag{7.41}$$

$$= 1 - P(\omega^*|\mathbf{x}) + P(\omega^*|\mathbf{x}) \sum_{\omega_i} P_D(\omega_i|\mathbf{x})$$

$$- \sum_{\omega_i} P(\omega_i|\mathbf{x})P_D(\omega_i|\mathbf{x}) \tag{7.42}$$

$$= 1 - P(\omega^*|\mathbf{x}) + \sum_{\omega_i} (P(\omega^*|\mathbf{x}) - P(\omega_i|\mathbf{x}))P_D(\omega_i|\mathbf{x}) \tag{7.43}$$

$$= 1 - P(\omega^*|\mathbf{x}) + (P(\omega^*|\mathbf{x}) - P(\hat{\omega^*}|\mathbf{x}))P_D(\hat{\omega^*}|\mathbf{x})$$

$$+ \sum_{\omega_i \neq \hat{\omega^*}} (P(\omega^*|\mathbf{x}) - P(\omega_i|\mathbf{x}))P_D(\omega_i|\mathbf{x}) \tag{7.44}$$

$$= \text{noise} + \text{bias} + \text{spread} \tag{7.45}$$

7.3.2.3 Domingos' Decomposition. Domingos suggests the following decomposition of the conditional error [235]

$$P(\text{error}|\mathbf{x}) = c_1 \times \text{noise} + \text{bias} + c_2 \times \text{variance} \tag{7.46}$$

where c_1 and c_2 are constants or expressions depending on which loss function we use. For the $0/1$ loss, both are expressions including the probabilities that participate

in the bias, variance, and noise themselves, which makes the relationship between the three components and the conditional probability nonlinear. Here we develop the decomposition for $0/1$ loss and two classes, ω_1 and ω_2. The conditional probability of error is

$$P(\text{error}|\mathbf{x}) = P(\omega_1|\mathbf{x})P_D(\omega_2|\mathbf{x}) + P(\omega_2|\mathbf{x})P_D(\omega_1|\mathbf{x}) \tag{7.47}$$

First consider the case of $bias = 0$. This means that either $\omega^* = \hat{\omega^*} = \omega_1$ or $\omega^* = \hat{\omega^*} = \omega_2$. In both cases Eq. (7.47) becomes

$$\begin{aligned}
P(\text{error}|\mathbf{x}) &= P(\omega^*|\mathbf{x})(1 - P_D(\hat{\omega^*}|\mathbf{x})) && (7.48)\\
&\quad + (1 - P(\omega^*|\mathbf{x}))P_D(\hat{\omega^*}|\mathbf{x}) && (7.49)\\
&= (1 - (1 - P(\omega^*|\mathbf{x})))(1 - P_D(\hat{\omega^*}|\mathbf{x}))\\
&\quad + (1 - P(\omega^*|\mathbf{x}))(1 - (1 - P_D(\hat{\omega^*}|\mathbf{x}))) && (7.50)\\
&= (1 - \text{noise}) \times \text{variance} + \text{noise} \times (1 - \text{variance}) && (7.51)\\
&= (1 - 2 \times \text{noise}) \times \text{variance} + \text{noise} && (7.52)
\end{aligned}$$

We note that the coefficient in front of the variance is nonnegative. This comes from the fact that the noise (Bayes error) is always less than a half. Indeed, the noise in this case is equal to the smallest of the two probabilities that sum to 1. Therefore, for unbiased $\mathbf{x}$ the error $P(\text{error}|\mathbf{x})$ decreases when the variance decreases. We can add a fictional bias term (zero in this case) to get the general form of the decomposition of the conditional error for *unbiased examples*

$$P(\text{error}|\mathbf{x}) = \text{bias} + (1 - 2 \times \text{noise}) \times \text{variance} + \text{noise} \tag{7.53}$$

Now consider the case of biased examples, that is, $bias = 1$. This means that if $\omega^* = \omega_1$ then $\hat{\omega^*} = \omega_2$ and vice versa. Then

$$\begin{aligned}
P(\text{error}|\mathbf{x}) &= P(\omega^*|\mathbf{x})P_D(\hat{\omega^*}|\mathbf{x}) + (1 - P(\omega^*|\mathbf{x}))(1 - P_D(\hat{\omega^*}|\mathbf{x})) && (7.54)\\
&= (1 - \text{noise}) \times (1 - \text{variance}) + \text{noise} \times \text{variance} && (7.55)\\
&= 1 + (2 \times \text{noise} - 1) \times \text{variance} - \text{noise} && (7.56)\\
&= \text{bias} + (2 \times \text{noise} - 1) \times \text{variance} - \text{noise} && (7.57)
\end{aligned}$$

The interesting fact here is that the coefficient in front of the variance is negative. This means that for *biased examples*, increasing the variance will bring the error down!

All decompositions of the error are aimed at studying the structure of the error for different classifier models and ensembles of classifiers. Suppose that we build our random classifier D using different data sets drawn from the distribution of the pro-

blem. It is natural to expect that simple classifiers such as the linear discriminant classifier will have high bias (deviation from the optimal model) and low variance. Conversely, flexible classifiers such as neural networks and decision trees will vary significantly from data set to data set because they will try to fit the particular realization of the data as close as possible. This means that they will have high variance but their bias will be low. If the classifier has a parameter that we can tune, then making the classifier more coarse and robust will diminish its sensitivity, therefore will decrease the variance but might increase the bias. This is referred to as the *bias-variance dilemma*. Different trends have been found in the literature. Sometimes varying a classifier parameter reduces both bias and variance, thereby giving a smaller error altogether. For example, increasing k in the k-nearest neighbor classifier is supposed to increase the bias and reduce the variance. Domingos found that for different data sets, one of the trends dominates the other and the total error might be steadily increasing with k or steadily decreasing with k [234]. For tree classifiers, the control parameter may be the depth of the tree or the constant used in prepruning. Typically, heavily pruned trees will have smaller variance and larger bias than trees fully grown to classify correctly all training samples.

In summary, bias is associated with underfitting the data, that is, the classifier cannot match well the optimal model; variance is associated with overfitting, that is, different optimal models are fitted on different data sets drawn from the distribution of the problem [233].

7.3.3 How Do Bagging and Boosting Affect Bias and Variance?

There is no general theory about the effects of bagging and boosting on bias and variance. The results of extensive experimental studies published recently can be summarized as follows. Bagging is assumed to reduce variance without changing the bias. However, it has been found to reduce the bias as well, for example, for high-bias classifiers. Boosting has different effects at its different stages. At its early iterations boosting primarily reduces bias while at the later iterations it has been found to reduce mainly variance [116,213,220,233,234,236,237].

Freund and Schapire [220] argue that bias-variance decomposition is not the appropriate analysis tool for boosting, especially boosting with reweighting. Their explanation of the success of AdaBoost is based on the margin theory. They acknowledge though that giving a theory about the randomization effects of bagging and boosting with resampling is an interesting open problem. Schapire et al. [213] state that it is unlikely that a "perfect" theory about voting methods exists, that is, a theory applicable to any base classifier and any source of independent identically distributed labeled data.

7.4 WHICH IS BETTER: BAGGING OR BOOSTING?

In principle, this is an ill-posed question. The old pattern recognition refrain springs to mind again: there is no "best" method for all problems. Many authors have

compared the two approaches including some large-scale experiments [106,116, 236,238,239]. The general consensus is that boosting reaches lower testing error. Boosting algorithms have been crowned as the "most accurate available off-the-shelf classifiers on a wide variety of data sets" [116]. However, it seems that they are sensitive to noise and outliers, especially for small data sets [116,236,239]. Variants of bagging, for example, random forests, have been found to be comparable to AdaBoost.

AdaBoost appears in two versions in the literature: with *resampling* and *reweighting*. In the resampling version the data set S_k for step k is obtained by subsampling with replacement from $\mathbf{Z}$. In the reweighting version, the classifier D_k takes into account the weights of the data points in some way. There is no strong evidence favoring one of the versions over the other [116,219,220].

APPENDIX 7A PROOF OF THE ERROR BOUND ON THE TRAINING SET FOR AdaBoost (TWO CLASSES)

Below we reproduce the proof of the upper bound on the training error of AdaBoost [219]. Our version differs from the original proof by the fact that we do not have separate steps for updating the weights and then updating the distribution. The proof is modified accordingly. The following Lemma is needed within the proof.

Lemma A.1. Let $a \geq 0$ and $r \in [0, 1]$. Then

$$a^r \leq 1 - (1 - a)r \tag{A.1}$$

Proof. The result of this lemma follows from the convexity property of functions. Let $p, q \in \Re$ and let $f(x)$ be a convex function $f : \Re \to \Re$. Then for any $t \in [0, 1]$,

$$f(tp + (1 - t)q) \leq tf(p) + (1 - t)f(q) \tag{A.2}$$

Geometrically, convexity means that for any point (x, y) on the line segment connecting two points, $(p, f(p))$ and $(q, f(q))$, on the graph of the function, $f(x) \leq y$.

A function f is convex if its second derivative is positive. Take a^r to be a function of r for a fixed $a \geq 0$. The second derivative

$$\frac{\partial^2(a^r)}{\partial r^2} = a^r(\ln a)^2 \tag{A.3}$$

is always nonnegative, therefore a^r is a convex function. The right-hand side of inequality (A.1) represents a point $(t = 1 - r)$ on the line through points (0, 1) and (1, a), both lying on the function graph (check by substitution). Therefore Eq. (A.1) holds for any $r \in [0, 1]$. ■

Figure 7A.1 shows two examples of a^r for $a = 15$ and $a = 0.02$.

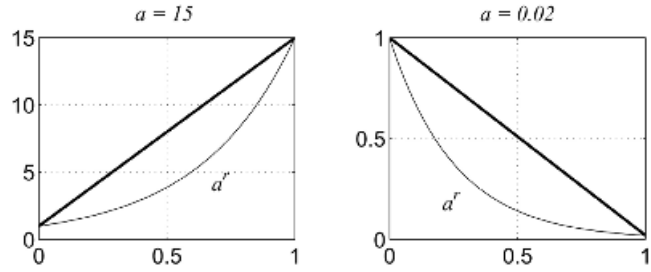


Fig. 7A.1 *Illustration of the convexity property of a^r for two values of a.*

Theorem 7.1. Let $\Omega = \{\omega_1, \omega_2\}$. Let ε be the ensemble training error and let ε_i, $i = 1, \ldots, L$, be the weighted training errors of the classifiers in $\mathcal{D}$ as in Eq. (7.11). Then

$$\varepsilon < 2^L \prod_{i=1}^{L} \sqrt{\varepsilon_i(1-\varepsilon_i)} \tag{A.4}$$

Proof. The proof consists of two parts. In Part 1, we prove a relationship between ε and β_i values calculated within the AdaBoost algorithm, and in Part 2, the upper bound is minimized by finding appropriate values of β_i. When substituting the optimal β_i in the bound found in Part 1, the thesis of the theorem (A.4) is proven.

Proof, Part 1. According to Eq. (7.11), the weighted training error of the first classifier is

$$\varepsilon_1 = \sum_{j=1}^{N} w_j^1 l_j^1 \tag{A.5}$$

where l_j^1 is the loss for object $\mathbf{z}_j$ in the training set $\mathbf{Z}$ due to classifier D_1. $l_j^1 = 1$ if $\mathbf{z}_j$ is misclassified and 0 otherwise. Subsequently we calculate β_1 from ε_1. Here we do not assume any particular expression for this relationship because our purpose is to derive β_i as a function of ε_i so as to minimize the ensemble error. The weights are updated to

$$w_j^2 = \frac{w_j^1 \beta_1^{(1-l_j^1)}}{\sum_{k=1}^{N} w_k^1 \beta_1^{(1-l_k^1)}} \tag{A.6}$$

Denote the normalizing coefficient at step i by

$$C_i = \sum_{k=1}^{N} w_k^i \beta_i^{(1-l_k^i)} \tag{A.7}$$

After we build and test the second classifier, we have

$$\varepsilon_2 = \sum_{j=1}^{N} w_j^2 l_j^2 \tag{A.8}$$

β_2 is calculated and the new weights are

$$w_j^3 = \frac{w_j^2 \beta_2^{(1-l_j^2)}}{\sum_{k=1}^{N} w_k^2 \beta_2^{(1-l_k^2)}} = \frac{w_j^1 \beta_1^{(1-l_j^1)} \beta_2^{(1-l_j^2)}}{C_1 C_2} \tag{A.9}$$

The general formula for the weights is then

$$w_j^{t+1} = w_j^1 \prod_{i=1}^{t} \frac{\beta_i^{(1-l_j^i)}}{C_i} \tag{A.10}$$

Denote by $\mathbf{Z}^{(-)}$ the subset of elements of $\mathbf{Z}$ that are misclassified by the ensemble. The ensemble error, weighted by the initial data weights w_j^1 is

$$\varepsilon = \sum_{\mathbf{z}_j \in \mathbf{Z}^{(-)}} w_j^1 \tag{A.11}$$

If we assign equal initial weights of $1/N$ to the objects, ε is the proportion of misclassifications on $\mathbf{Z}$ made by the ensemble.

Since at each step, the sum of the weights in our algorithm equals one,

$$1 = \sum_{j=1}^{N} w_j^{L+1} \quad \geq \quad \sum_{\mathbf{z}_j \in \mathbf{Z}^{(-)}} w_j^{L+1} = \sum_{\mathbf{z}_j \in \mathbf{Z}^{(-)}} w_j^1 \prod_{i=1}^{L} \frac{\beta_i^{(1-l_j^i)}}{C_i} \tag{A.12}$$

For the ensemble to commit an error in the labeling of some $\mathbf{z}_j$, the sum of the weighted votes for the wrong class label in Eq. (7.14) must be larger than the sum for the correct label. Recall that l_j^i is 1 if D_i misclassifies $\mathbf{z}_j$. Then

$$\sum_{i=1}^{L} l_j^i \ln\left(\frac{1}{\beta_i}\right) \geq \sum_{i=1}^{L} (1 - l_j^i) \ln\left(\frac{1}{\beta_i}\right) \tag{A.13}$$

Taking exponent on both sides,

$$\prod_{i=1}^{L} \beta_i^{-l_j^i} \geq \prod_{i=1}^{L} \beta_i^{-(1-l_j^i)} \tag{A.14}$$

Multiplying by $\prod_i \beta_i$ on both sides ($\prod_i \beta_i > 0$),

$$\prod_{i=1}^{L} \beta_i^{1-l_j^i} \geq \prod_{i=1}^{L} \beta_i \prod_{i=1}^{L} \beta_i^{-(1-l_j^i)} \tag{A.15}$$

Since β_i is always positive,

$$\prod_{i=1}^{L} \beta_i^{2(1-l_j^i)} \geq \prod_{i=1}^{L} \beta_i \tag{A.16}$$

Taking square root on both sides,

$$\prod_{i=1}^{L} \beta_i^{(1-l_j^i)} \geq \prod_{i=1}^{L} \beta_i^{1/2} \tag{A.17}$$

Taking Eqs. (A.12), (A.17), and (A.11) together,

$$1 \geq \sum_{\mathbf{z}_j \in \mathbf{Z}^{(-)}} w_j^1 \prod_{i=1}^{L} \frac{\beta_i^{(1-l_j^i)}}{C_i} \tag{A.18}$$

$$\geq \left(\sum_{\mathbf{z}_j \in \mathbf{Z}^{(-)}} w_j^1 \right) \prod_{i=1}^{L} \frac{\beta_i^{\frac{1}{2}}}{C_i} = \varepsilon \cdot \prod_{i=1}^{L} \frac{\beta_i^{1/2}}{C_i} \tag{A.19}$$

Solving for ε,

$$\varepsilon \leq \prod_{i=1}^{L} \frac{C_i}{\beta_i^{1/2}} \tag{A.20}$$

From the Lemma,

$$C_i = \sum_{k=1}^{N} w_k^i \beta_i^{(1-l_k^i)} \leq \sum_{k=1}^{N} w_k^i (1 - (1 - \beta_i)(1 - l_k^i)) \tag{A.21}$$

$$= \sum_{k=1}^{N} w_k^i (\beta_i + l_k^i - \beta_i l_k^i) \tag{A.22}$$

$$= \beta_i \sum_{k=1}^{N} w_k^i + \sum_{k=1}^{N} w_k^i l_k^i - \beta_i \sum_{k=1}^{N} w_k^i l_k^i \tag{A.23}$$

$$= \beta_i + \varepsilon_i - \beta_i \varepsilon_i = 1 - (1 - \beta_i)(1 - \varepsilon_i) \tag{A.24}$$

Combining Eqs. (A.20) and (A.24)

$$\varepsilon \leq \prod_{i=1}^{L} \frac{1-(1-\beta_i)(1-\varepsilon_i)}{\sqrt{\beta_i}} \tag{A.25}$$

Proof, Part 2. The next step is to find β_i values that minimize the bound of ε in Eq. (A.25).

Denote the right-hand side of Eq. (A.25) by $\varepsilon_{\max}$. The first derivative with respect to β_i is

$$\frac{\partial \varepsilon_{\max}}{\partial \beta_i} = \frac{\beta_i(1-\varepsilon_i)-\varepsilon_i}{2\beta_i\sqrt{\beta_i}} \times (\text{a constant}) \tag{A.26}$$

Setting $\partial \varepsilon_{\max}/\partial \beta_i = 0$ and solving for β_i, we obtain

$$\beta_i = \frac{\varepsilon_i}{1-\varepsilon_i} \tag{A.27}$$

The second derivative of $\varepsilon_{\max}$ at $\beta_i = \varepsilon_i/1-\varepsilon_i$ is

$$\frac{\partial^2 \varepsilon_{\max}}{\partial \beta_i^2} = (1-\varepsilon_i)\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)^{3/2} \times (\text{a constant}) > 0 \tag{A.28}$$

It is straightforward to verify that the constant is positive at $\beta_i = \varepsilon_i/1-\varepsilon_i$, $i = 1, \ldots, L$. Thus the solution for β_i is a minimum of $\varepsilon_{\max}$.

Substituting Eq. (A.27) into Eq. (A.25) leads to the thesis of the theorem

$$\varepsilon < 2^L \prod_{i=1}^{L} \sqrt{\varepsilon_i(1-\varepsilon_i)} \tag{A.29}$$

■

APPENDIX 7B PROOF OF THE ERROR BOUND ON THE TRAINING SET FOR AdaBoost (*C* CLASSES)

Theorem 7.2. Let $\Omega = \{\omega_1, \ldots, \omega_c\}$. Let ε be the ensemble training error and let ε_i, $i = 1, \ldots, L$, be the weighted training errors of the classifiers in $\mathcal{D}$ as in Eq. (7.11) and $\varepsilon_i < \frac{1}{2}$. Then

$$\varepsilon < 2^L \prod_{i=1}^{L} \sqrt{\varepsilon_i(1-\varepsilon_i)} \tag{B.1}$$

Proof. The difference between the proof of this theorem and the proof of Theorem 7.1 will appear in the inequality for β_i (A.13). Making a classification error with more than two classes means that the score for the wrong label is higher than any other score, including that of the right class label. Let us split the set of L classifiers into three subsets according to their outputs for a particular $\mathbf{z}_j \in \mathbf{Z}$:

$\mathcal{D}^w \subset \mathcal{D}$, the set of classifiers whose output is the winning (wrong) label;

$\mathcal{D}^+ \subset \mathcal{D}$, the set of classifiers whose output is the true label;

$\mathcal{D}^- \subset \mathcal{D}$, the set of classifiers whose output is another (wrong) label.

The support for the winning class is

$$\sum_{D_i \in \mathcal{D}^w} \ln\left(\frac{1}{\beta_i}\right) \geq \sum_{D_i \in \mathcal{D}^+} \ln\left(\frac{1}{\beta_i}\right) \tag{B.2}$$

Add on both sides $\sum_{D_i \in \mathcal{D}^w}(\cdot) + \sum_{D_i \in \mathcal{D}^-}(\cdot)$ to obtain

$$2\sum_{D_i \in \mathcal{D}^w} \ln\left(\frac{1}{\beta_i}\right) + \sum_{D_i \in \mathcal{D}^-} \ln\left(\frac{1}{\beta_i}\right) \geq \sum_{i=1}^{L} \ln\left(\frac{1}{\beta_i}\right) \tag{B.3}$$

To arrive at Eq. (A.17), we need to add $\sum_{D_i \in \mathcal{D}^-}(\cdot)$ on the left side of the inequality. This is only possible if the added quantity is positive. To guarantee this, we require that all the terms in the summation are positive, that is, $\ln(1/\beta_i) \geq 0$, which is equivalent to

$$\beta_i \leq 1 \tag{B.4}$$

This requirement has to be enforced for all β_i values because different classifiers might be in $\mathcal{D}^-$ for different $\mathbf{z}_j \in \mathbf{Z}$, and Eq. (B.3) must hold for any $\mathbf{x}$ drawn from the distribution of the problem.

Then the left-hand side of Eq. (B.3) is twice the sum of all weights for the wrong classes; that is,

$$2\sum_{i=1}^{L} l_j^i \ln\left(\frac{1}{\beta_i}\right) \geq \sum_{i=1}^{L} \ln\left(\frac{1}{\beta_i}\right) \tag{B.5}$$

whereby Eq. (A.17) is derived.

The rest of the proof is exactly as in Theorem 1. ■

Taking the expression for the optimal β_i (A.27), and solving for $\beta_i < 1$ leads to

$$\varepsilon_i \leq 0.5$$

AdaBoost.M1 in Figure 7.7 satisfies this requirement by ignoring all classifiers whose weighted error is greater than 0.5.

8

Miscellanea

8.1 FEATURE SELECTION

The classifiers in the ensemble can be built on different subsets of features, either disjoint or overlapping. Feature selection aims at a more efficient computation and a higher accuracy of the ensemble.

8.1.1 Natural Grouping

In some problems the features are naturally grouped. For example, in text-independent speaker identification, different groups of features are related to the pitch of the signal, and the speech spectrum. The speech spectrum can be further characterized by the linear predictive coefficients, the cepstrum, and so on [240]. In handwritten digit recognition, an image can be viewed from different perspectives, for example, pixels, morphological features, Fourier coefficients of the character shapes, and so on [157,241]. Sometimes the groups of features are measured at different geographical locations, for example, radar images of a flying object. Instead of transmitting all the features and making a decision centrally, individual classifiers can be built and only their decisions will have to be transmitted.

8.1.2 Random Selection

Choosing random subsets of features is termed the *random subspace method* [218,242,243]. Each classifier in the ensemble is built upon a randomly chosen subset of features of predefined size d. Ho [243] suggests that good results are obtained

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

for tree classifiers built upon $d \approx n/2$ features, where n is the total number of features. The random subspace method has been found to work well when there is redundant information which is "dispersed" across all the features rather than concentrated in a subset of them [218,243].

Pękalska et al. [244] consider a *dissimilarity representation* of the objects. Each of the N objects is described by N features, which are the distances to all the objects (including itself). Such representation is useful when the data originally contains a prohibitive number of features or when a similarity measure between objects is easily available. Pękalska et al. proposed using a linear discriminant classifier on randomly selected feature subsets and recommend d between 4 and 30 percent of the (similarity) features.

Latinne et al. [21] propose combining bagging with random selection. B bootstrap replicates are sampled from the training set $\mathbf{Z}$ and for each replicate R subsets of features are chosen. The proportion of features, K, is the third parameter of the algorithm. The ensemble consists of $L = B \times R$ classifiers. The combination between bagging and feature selection aims at making the ensemble more diverse than when using either of the methods alone. Latinne et al. show that the combined technology outperforms each of the single ones.

We can apply various heuristic search techniques such as genetic algorithms, tabu search, and simulated annealing for subset selection [245,246]. The feature subsets can be selected one at a time or all at the same time. When one subset is selected for each run of the algorithm, we must ensure that the next run takes into account all the subsets selected hitherto. Alternatively, all feature subsets can be derived in one run of the algorithm by optimizing some ensemble performance criterion function.

8.1.3 Nonrandom Selection

8.1.3.1 The "Favourite Class" Model. Oza and Tumer [247] suggest a simple algorithm for selecting feature subsets that they call *input decimation*. The ensemble consists of $L = c$ classifiers where c is the number of classes. Each classifier has a "favorite" class. To find the feature subset for the classifier D_i with favorite class ω_i, we calculate the correlation between each feature and the class label variable. The class label variable has value 0 for all objects that are not in class ω_i and 1 for all objects that are in class ω_i.

Example: Correlation Between a Feature and a Class Label Variable. Suppose the data set consists of nine objects labeled in three classes. The values of feature x_1 and the class labels for the objects are as follows:

Object	$\mathbf{z}_1$	$\mathbf{z}_2$	$\mathbf{z}_3$	$\mathbf{z}_4$	$\mathbf{z}_5$	$\mathbf{z}_6$	$\mathbf{z}_7$	$\mathbf{z}_8$	$\mathbf{z}_9$
x_1	2	1	0	5	4	8	4	9	3
Class	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_3	ω_3	ω_3

The class label variable for ω_1 has values [1, 1, 1, 0, 0, 0, 0, 0, 0] for the nine objects. Its correlation with x_1 is -0.75.

The n correlations are sorted by absolute value and the features corresponding to the n_i largest correlations are chosen as the subset for classifier D_i. Note that ω_i is only a favorite class to D_i but D_i is trained to recognize all the c classes. Selecting the subsets in this way creates diversity within the ensemble. Even with this simple selection procedure, the ensemble demonstrated better performance than the random subset selection method [247].

There are numerous feature selection methods and techniques that can be used instead of the sorted correlations. Such are the methods from the sequential group (forward and backward selection) [1,248], the floating selection methods [249], and so on. Given that the number of subsets needed is the same as the number of classes, using a more sophisticated feature selection technique will not be too computationally expensive and will ensure higher quality of the selected feature subset.

8.1.3.2 The Iterative Model. Another "favorite class" feature selection method is proposed by Puuronen et al. [250]. They devise various criteria instead of the simple correlation with the class variable and suggest an iterative procedure by which the selected subsets are updated. The procedure consists of the following general steps:

1. Generate an initial ensemble of c classifiers according to the "favorite class" procedure based on simple correlation.
2. Identify the classifier whose output differs the least from the outputs of the other classifiers. We shall call this the *median* classifier. The median classifier is identified using some pairwise measure of diversity, which we will denote by $\Delta(D_i, D_j)$. High values of Δ will denote large disagreement between the outputs of classifiers D_i and D_j. $\Delta(D_i, D_j) = 0$ means that D_i and D_j produce identical outputs.[32] The median classifier D_k is found as

$$D_k = \arg\min_i \sum_{j=1}^{L} \Delta(D_i, D_j) \tag{8.1}$$

3. Take the feature subset used to create D_k. Altering the present/absent status of each feature, one at a time, produce n classifier-candidates to replace the median classifier. For example, let $n = 4$ and let D_k be built on features x_1 and x_3. We can represent this set as the binary mask $[1, 0, 1, 0]$. The classifier-candidates to replace D_k will use the following subsets of features: $[0, 0, 1, 0]$ (x_3), $[1, 1, 1, 0]$ (x_1, x_2, x_3), $[1, 0, 0, 0]$ (x_1) and $[1, 0, 1, 1]$ (x_1, x_3, x_4). Calculate the ensemble accuracy with each replacement. If there is an improvement, then keep the replacement with the highest improvement, dismiss the other candidate classifiers, and continue from step 2.
4. If no improvement is found at step 3, then stop and return the current ensemble.

[32] Diversity will be discussed in detail in Chapter 10. For now, we can think of Δ as the proportion of all objects for which one of the two classifiers is correct and the other one is wrong.

This greedy algorithm has been shown experimentally to converge quickly and to improve upon the initial ensemble. Numerous variants of this simple iterative procedure can be designed. First, there is no need to select the initial ensemble according to the "favorite class" procedure. Any ensemble size and any initial subset of features might be used. In the iterative algorithm in Ref. [251], the initial ensemble is generated through the random subspace method introduced above. Another possible variation of the procedure, as suggested in Ref. [251] is to check all the ensemble members, not only the median classifier. In this case, the task of calculating the ensemble accuracy for each classifier candidate for replacement might become computationally prohibitive. Therefore, the authors suggest that the criterion for accepting the replacement should be the individual accuracy of the classifier plus the classifier's diversity weighted by a coefficient α. The (individual) diversity of classifier D_i is $\sum_j \Delta(D_i, D_j)$. Thus diversity is enforced in the ensemble. The authors note that using only accuracy is insufficient for improvement on the initial ensemble. If the *ensemble* accuracy was used again as the criterion for accepting the replacement, then diversity would not be needed as we optimize directly our target function.

Another detail of the iterative method that can be varied is whether we accept the changes gradually as they occur during generation of the candidate classifiers, or we keep track of the improvement and accept only one replacement classifier after a whole cycle through possible candidates. There is no evidence that favors either of the two strategies for selecting a replacement classifier.

8.1.3.3 The Incremental Model. Günter and Bunke [252] propose a method for creating classifier ensembles based on feature subsets. Their ensemble is built gradually, one classifier at a time, so that the feature *subsets* selected for the previous classifiers are not allowed for the subsequent classifiers. However, intersection of the subsets is allowed. The authors suggest that any feature selection method could be used and advocate the floating search for being both robust and computationally reasonable. There are various ways in which the ban on the previous subsets can be implemented. Günter and Bunke use different feature selection algorithms relying on their suboptimality to produce different feature subsets. To evaluate a subset of features S, they use the ensemble performance rather than the performance of the individual classifier built on the subset of features. This performance criterion, on its own, will stimulate diversity in the ensemble. It is not clear how successful such heuristics are in the general case but the idea of building the ensemble incrementally by varying the feature subsets is certainly worthy of a deeper look.

In the case of a very large amount of features, we can afford to ban completely the features already used by the previous classifiers. Thus the classifiers will use disjoint feature subsets.

8.1.4 Genetic Algorithms

Genetic algorithms (GA) offer a guided random search in the space of all possible feature subsets. Depending on the encoding, there are two general ways to run a GA for selecting the L feature subsets.

8.1.4.1 Approach A. We can try to select the L subsets individually. GAs operate on a population of individuals. Each subset will then be a member of the population. The GA is aiming at finding a single individual of the highest possible quality measured by its *fitness*. The aim in creating an ensemble is not finding one best classifier but finding classifiers that will be jointly better than the single best individual. To achieve this, the individual members of the population have to be both accurate and diverse. While accuracy is accounted for by the fitness function, diversity is not. On the contrary, a GA will converge to a single solution and all the members of the population will be clones of the same individual. Hence there should be a mechanism for maintaining diversity and stopping the evolution process when a good ensemble is reached. Diversity of the feature subsets (the genotype) does not guarantee diversity of the classifier outputs (the phenotype). Nonetheless, diversity in the genotype is the easily available option within this approach. The deviation of a feature subset from the remaining members of the population is a possible measure that can be used together with the accuracy of the classifier built on this feature subset.

Example: Diversity in a Population of Feature Subsets. The most convenient way of representing a feature subset is by a binary vector of length n. The ith bit of the vector is 1 if the ith feature is included in the subset and 0 otherwise. A GA operates on a set of, say, M such binary vectors, called *chromosomes*. The population of chromosomes is evolved by producing offspring and keeping the fittest M individuals as the next generation.

Let $X = \{x_1, \ldots, x_{10}\}$ be a set of features. A population of six chromosomes (individuals), $S_1, \ldots, S_6$, is displayed below

S_1	0	1	0	1	0	0	0	1	0	0	$\{x_2, x_4, x_8\}$
S_2	1	1	1	0	0	0	0	0	0	1	$\{x_1, x_2, x_3, x_{10}\}$
S_3	0	0	1	1	1	1	0	0	0	0	$\{x_3, x_4, x_5, x_6\}$
S_4	0	0	0	0	0	0	0	1	0	1	$\{x_8, x_{10}\}$
S_5	0	1	1	0	0	0	0	0	0	0	$\{x_2, x_2\}$
S_6	1	1	0	1	1	1	0	0	1	1	$\{x_1, x_2, x_4, x_5, x_6, x_9, x_{10}\}$

The simplest way of calculating how different subset S_i is from the remaining subsets in the population is to use an average measure, for example, the averaged Hamming distance between S_i and each of the other $M - 1$ subsets. For the above example, the five distances for S_1 are 5, 5, 3, 3, and 6, respectively. Then the diversity of S_1 can be calculated as $d_1 = \frac{5+5+3+3+6}{5} = 4.4$. Equivalently, d_i values can be calculated as the sum of the absolute difference between bit k of chromosome S_i and the averaged chromosome $\bar{S} = \frac{1}{5}\sum S_i$. The individual diversities of the other subsets are $d_2 = 4.4$, $d_3 = 5.2$, $d_4 = 4.8$, $d_5 = 4.0$, and $d_6 = 6.0$. Suitably weighted, these diversity values can be taken together with the respective accuracies as the fitness value.

The problem with this approach is that the *ensemble* accuracy does not participate in the evolution process at any stage. It is not clear whether the ensemble will be any

better than the fittest chromosome. The advantage of this approach to the second approach detailed below is the lower computational cost.

Genetic algorithms for feature selection according to approach A are suggested in Refs. [245] and [253].

8.1.4.2 Approach B. In this approach, each individual in the population represents the entire ensemble. We can use disjoint subsets (Approach B.1) or allow for intersection (Approach B.2).

B.1. To represent disjoint subsets, we can keep the length of the vector (the chromosome) at n and use integers from 0 to L. The value at position i will denote which classifier uses feature x_i; zero will mean that feature x_i is not used at all. An integer-valued GA can be used to evolve a population of ensembles. The fitness of a chromosome can be directly a measure of the accuracy of the ensemble the chromosome represents. For example, let X be a set of ten features. Consider an ensemble of four classifiers. A chromosome [1, 1, 4, 0, 3, 3, 1, 2, 3, 3] will denote an ensemble $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$ where D_1 uses features x_1, x_2, and x_7; D_2 uses x_8; D_3 uses x_5, x_6, x_9, and x_{10}; and D_4 uses x_3. Feature x_4 is not used by any of the classifiers.

B.2. To allow any subset of features to be picked by any classifier, the ensemble can be represented by a binary chromosome of length $L \times n$. The first n bits will represent the feature subset for classifier D_1, followed by the n bits for classifier D_2, and so on. A standard binary valued GA can be used with this representation.

Approach B is more computationally demanding than approach A but the accuracy of the ensemble can be calculated right away. Approach B.1 is simpler than B.2, but the requirement that the ensemble members use disjoint feature subsets might be too restrictive.

In all GA implementations, special care should be taken to make sure that only viable chromosomes are produced. For example, in approach B.1, it might happen that not all L classifiers receive subsets of features. Then the ensemble will have fewer members than intended. If we insist on the number of classifiers, L, we might wish to kill any chromosome that does not contain all the integers from 1 to L. Alternatively, the algorithm can be left to pick the number of classifiers (up to L) itself by guarding only against a chromosome containing only zeros.

Examples of GAs within approaches B.1 and B.2 can be found in Ref. [246].

8.1.5 Ensemble Methods for Feature Selection

Suppose we *reverse* the problem: instead of selecting features for building an ensemble we use ensemble technologies to select features. A variant of AdaBoost for feature selection for two-class problems is proposed in Ref. [230]. Decision stumps are used as the base classifiers so that each tree in the ensemble consists of a root and two leaves. The split at the root is done on a single feature that is *different* for each classifier. The AdaBoost has been modified for this application so as to remove

a feature from the possible set of candidate features as soon as that feature has been used. The resultant ensemble uses a collection of features optimized for combined performance. Long and Vega [230] propose this method for microarray data where the number of features (genes) is large and the number of cases (tissue samples) is comparatively small. The classes could be, for example, tumor and normal tissue. Feature selection for microarray analysis is important for at least two reasons. First, finding a small subset of important genes might trigger new research directions and second, the diagnosis of the two classes becomes cheaper as only few of possibly thousand features have to be measured.

Practically, an ensemble of decision stumps is a simple neural network like architecture as shown in Figure 8.1. The main asset of the model is that once the ensemble size L is specified, the training becomes completely automatic. The choice of L is not crucial anyway. Ensembles of 50 classifiers are usually sufficient. The value of L might be specified as the desired number of features for the respective application (genes in microarray analysis).

It is not clear how this method for feature selection compares to the other existing methods. The fact that it is tied with AdaBoost, the "best classifier off-the-shelf" [116], in an automatic training procedure suggests that the method could be a good competitor.

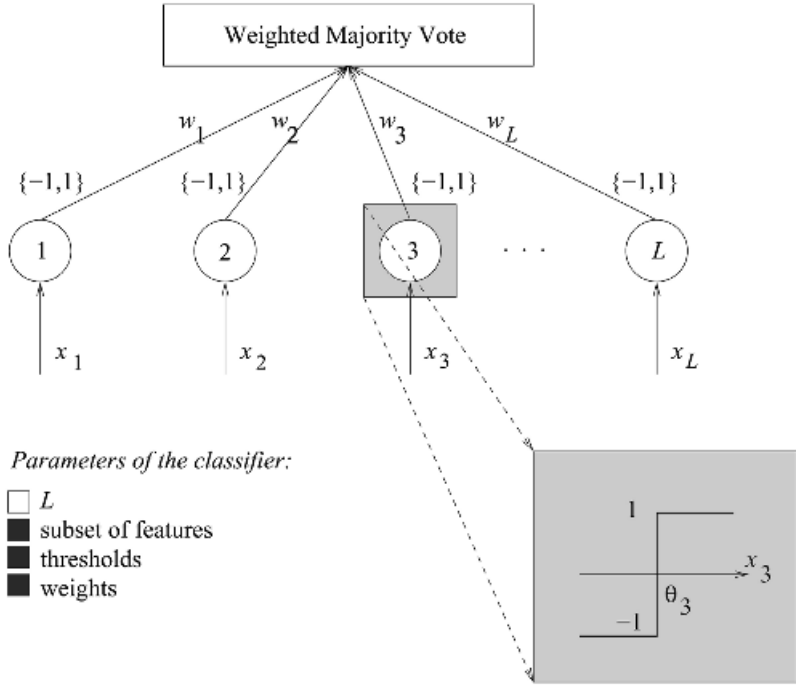


Fig. 8.1 *A classifier ensemble consisting of decision stumps. All the parameters of the ensemble are tuned by AdaBoost except the number of classifiers, L, which has to be specified in advance.*

8.2 ERROR CORRECTING OUTPUT CODES

The error correcting output codes (ECOC) ensemble strategy is developed for problems with multiple classes [254–259]. The idea is to avoid solving the multiclass problem directly and to break it into dichotomies instead. Each classifier in the ensemble discriminates between two possibly compound classes. For example, let $\Omega = \{\omega_1, \ldots, \omega_{10}\}$. We can break Ω into $\Omega = \{\Omega^{(1)}, \Omega^{(2)}\}$ where $\Omega^{(1)} = \{\omega_1, \ldots, \omega_5\}$ and $\Omega^{(2)} = \{\omega_6, \ldots, \omega_{10}\}$.

8.2.1 Code Designs

8.2.1.1 The Code Matrix. We can represent each split of the set of c classes as a binary vector of length c with 1s for the classes in $\Omega^{(1)}$ and 0s for the classes in $\Omega^{(2)}$. The corresponding vector for the above example is $[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]^T$. The set of all such vectors has 2^c elements. However, not all of them correspond to different splits. Consider $[0, 0, 0, 0, 0, 1, 1, 1, 1, 1]^T$. Even if the Hamming distance between the two binary vectors is equal to the maximum possible value 10, the two subsets are again $\Omega^{(1)}$ and $\Omega^{(2)}$, only with swapped labels. Since there are two copies of each split within the total of 2^c splits, the number of different splits is $2^{(c-1)}$. The splits $\{\Omega, \emptyset\}$ and the corresponding $\{\emptyset, \Omega\}$ are of no use because they do not represent any discrimination task. Therefore the number of possible *different* splits of a set of c class labels into two nonempty disjoint subsets (dichotomies) is

$$2^{(c-1)} - 1 \tag{8.2}$$

We can choose L dichotomies to be the classifier assignments. These can be represented as a binary *code matrix* C of size $c \times L$. The (i, j)th entry of C, denoted $C(i, j)$ is 1 if class ω_i is in $\Omega_j^{(1)}$ or 0, if class ω_i is in $\Omega_j^{(2)}$. Thus each row of the code matrix, called a *codeword*, corresponds to a class and each column corresponds to a classifier. Below is a code matrix for $c = 4$ classes with all possible $2^{(4-1)} - 1 = 7$ *different* dichotomies.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
ω_1	0	0	0	1	0	1	1
ω_2	0	0	1	0	0	0	0
ω_3	0	1	0	0	1	0	1
ω_4	1	0	0	0	1	1	0

Suppose that the classifiers in the ensemble output binary labels $(s_1, \ldots, s_L)$ for a given input $\mathbf{x}$. The Hamming distance between the classifier outputs and the codewords for the classes is calculated and the class with the shortest Ham-

ming distance is chosen as the label of $\mathbf{x}$. For the above example, let $(s_1, \ldots, s_L) = (0, 1, 1, 0, 1, 0, 1)$. The Hamming distances to the class codewords are 5, 3, 1, and 5, respectively, so label ω_3 is assigned to $\mathbf{x}$. The support for class ω_j can be expressed as

$$\mu_j(\mathbf{x}) = -\sum_{i=1}^{L} |s_i - C(j, i)| \tag{8.3}$$

8.2.1.2 Row and Column Separation for Error Correcting Output Codes. To take the most advantage of an ECOC classifier, the code matrix should be built according to two main criteria.

Row Separation. In order to avoid misclassifications, the codewords should be as far apart from one another as possible. We can still recover the correct label for $\mathbf{x}$ even if several classifiers have guessed wrongly. A measure of the quality of an error correcting code is the minimum Hamming distance between any pair of codewords. If this distance is denoted by H_c, the number of errors that the code is guaranteed to be able to correct is

$$\left\lfloor \frac{H_c - 1}{2} \right\rfloor \tag{8.4}$$

Column Separation. It is important that the dichotomies given as the assignments to the ensemble members are as different from each other as possible too. This will ensure low correlation between the classification errors and will increase the ensemble accuracy [256]. The distance between the columns must be maximized keeping in mind that the complement of a column gives the same split of the set of classes. Therefore, the column separation should be sought by maximizing

$$H_L = \min_{i,j,i \neq j} \min \left\{ \sum_{k=1}^{c} |C(k, i) - C(k, j)|, \sum_{k=1}^{c} |1 - C(k, i) - C(k, j)| \right\},$$
$$i, j \in \{1, 2, \ldots, L\} \tag{8.5}$$

8.2.1.3 Generation Methods for Error Correcting Output Codes. Below we explain three simple ECOC generation methods.

One-per-Class. The standard ECOC is the so-called "one-per-class" code, which is the default target output for training neural network classifiers for multiple classes. The target function for class ω_j is a codeword containing 1 at position j and 0s elsewhere. Thus the code matrix is the identity matrix of size c and we only build $L = c$ classifiers. This encoding is of low quality because the Hamming distance between any two rows is 2, and so the error correcting power is $\left\lfloor \frac{2-1}{2} \right\rfloor = 0$.

Exhaustive Codes. Dietterich and Bakiri [256] give the following procedure for generating all possible $2^{(c-1)} - 1$ different classifier assignments for c classes. They suggest that exhaustive codes should be used for $3 \leq c \leq 7$.

1. Row 1 is all ones.
2. Row 2 consists of $2^{(c-2)}$ zeros followed by $2^{(c-2)} - 1$ ones.
3. Row 3 consists of $2^{(c-3)}$ zeros, followed by $2^{(c-3)}$ ones, followed by $2^{(c-3)}$ zeros, followed by $2^{(c-3)} - 1$ ones.
4. In row i, there are alternating $2^{(c-i)}$ zeros and ones.
5. The last row is 0, 1, 0, 1, 0, 1, . . . , 0.

The exhaustive code for $c = 4$ obtained through this procedure is as follows:

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
ω_1	1	1	1	1	1	1	1
ω_2	0	0	0	0	1	1	1
ω_3	0	0	1	1	0	0	1
ω_4	0	1	0	1	0	1	0

For $8 \leq c \leq 11$ Dietterich and Bakiri [256] suggest to select columns from the exhaustive code by an optimization procedure. Note that for $c = 3$ the exhaustive code will be the same as the one-per-class code, which shows that problems with a small number of classes might not benefit from the ECOC approach. For values of c larger than 11, random code generation is recommended.

A Matlab code for generating an exhaustive code for a given c is provided in Appendix 8A.

Random Generation. Authors of studies on ECOC ensembles share the opinion that random generation of the codewords is a reasonably good method [256,259]. Although these studies admit that more sophisticated procedures might lead to better codes, they also state that the improvement in the code might have only marginal effect on the ensemble accuracy. The example below illustrates the random ECOC generation.

Example: Random Error Correcting Output Code Generation. In this method each bit in each codeword is set to either 0 or 1 with probability 0.5. The best code matrix is retained out of T random generations, where T is a prespecified constant. Denote by H_c the minimum Hamming distance between the codewords of the c classes and by H_L the minimum Hamming distance between the columns of the code matrix (the L dichotomies for the classifiers). The criterion that we want to maximize is the sum $H_c + H_L$.

We ran the ECOC generating procedure for all combinations of number of classes $c = \{4, 6, 8, \ldots, 16\}$ and $L = \{5, 10, 15, \ldots, 30\}$. The minimum Hamming

TABLE 8.1 Minimum Hamming Distances H_c and H_L for ECOCs for Different Values of c and L.

c	$L \rightarrow 5$	10	15	20	25	30
(a) H_c (minimum distance between codewords)						
4	2	5	8	11	14	16
6	2	5	7	9	12	14
8	1	4	6	8	10	12
10	1	3	6	7	10	12
12	1	3	5	7	9	11
14	1	3	5	6	9	11
16	0	2	4	6	8	10
(b) H_L (minimum distance between classifier assignments)						
4	1	0	0	0	0	0
6	2	1	1	0	0	0
8	4	2	1	1	1	1
10	4	2	2	1	1	1
12	5	3	3	2	2	2
14	6	4	3	3	3	3
16	7	5	4	4	3	3

distances between the codewords, H_c, for all combinations of c and L are given in Table 8.1a, and the minimum Hamming distances H_L are given in Table 8.1b.

Naturally, the largest distance between the codewords is found for the codes with the largest length. However, for these codes, we cannot arrange for good separability between columns. For example, with $L = 30$ classifiers (30 bits in the codeword), the best code found by the random search for $T = 500$ had a minimum Hamming distance between the codewords of $H_c = 14$ but there are classifiers that must solve exactly the same dichotomy ($H_L = 0$).

The code for the randomly generated ECOCs is given in Appendix 8B.

8.2.2 Implementation Issues

The standard example of an ECOC classifier is a neural network (NN) trained with the one-per-class model for which $L = c$. For example, in a 10-class problem, the NN target for class ω_3 would be $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]^T$. Instead of this vector we can supply the L-bit codeword for class ω_3. Thus a single NN will solve the whole problem.

Figure 8.2 shows three possible implementations of a classifier for a multiclass problem. Subplot (a) is the standard classifier, called "direct multiclass representation" [256] whereby the output is a class label. We can place in this group the NN with one-per-class output and the winner-takes-all principle for inferring the class label of $\mathbf{x}$. Subplot (b) shows a "monolithic" system as termed in Ref. [260]. Here one classifier, for example, an NN, learns all the bits of the codewords. The monolithic NN approach has been found to lead to a lower error rate than the direct multiclass representation [256]. Subplot (c) shows a system called a "parallel dichot-

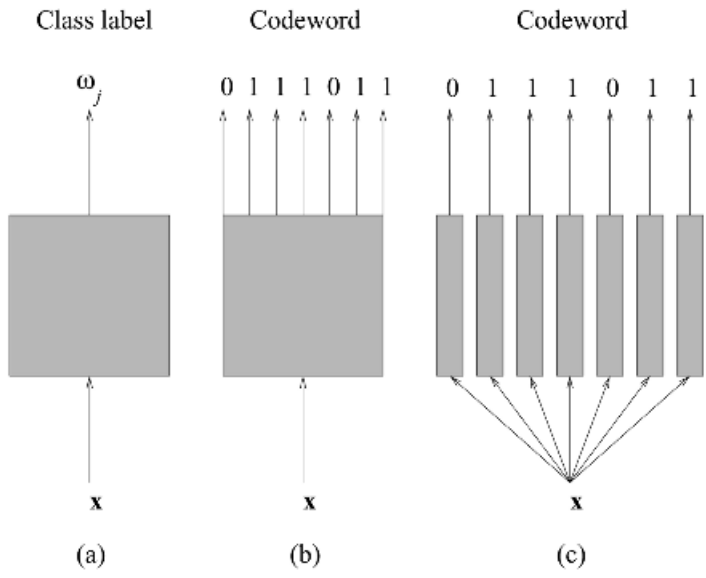


Fig. 8.2 *Three implementations of a classifier for a multiclass problem.*

omizer" [260]. In this model each dichotomy is learned by a separate classifier. The parallel dichotomizer has been found to be better than the monolithic system [260]. Even if the training process might take longer and the computation of the class label might be more time consuming, when accuracy has the highest priority, the parallel approach should be preferred.

A curious difference between a parallel ECOC system and a standard classifier ensemble is that each classifier in the ECOC system solves a different two-class problem whereas in standard classifier ensembles all classifiers solve the same (possibly multiclass) problem [256].

8.2.3 Error Correcting Ouput Codes, Voting, and Decision Templates

The combiner of an ECOC ensemble is the minimum Hamming distance. This can be viewed as majority voting as follows. Suppose that classifier D_i solves the dichotomy $\{\Omega_i^{(1)}, \Omega_i^{(2)}\}$. Let the decision of D_i be $s_i = 1$, that is, compound class $\Omega_i^{(1)}$ is chosen. Each individual class within $\Omega_i^{(1)}$ will obtain one vote from D_i. Since each dichotomy contains all the classes, each class will obtain a vote (for or against) from each classifier. Selecting the class with the largest sum of votes is equivalent to making a decision in favor of the class whose codeword has the lowest Hamming distance to the binary word of the L outputs $s_1, \ldots, s_L$. If the classifiers are made to learn very different boundaries, then there is a good chance that their errors will be unrelated.

TABLE 8.2 Code Matrix for $c = 8$ Classes and $L = 15$ Dichotomizers.

ω_1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0
ω_2	1	0	0	1	1	1	0	0	1	1	1	1	0	1	0
ω_3	1	0	0	0	1	1	0	1	1	0	0	1	0	0	0
ω_4	0	1	0	0	1	1	0	1	1	0	1	0	1	0	0
ω_5	1	1	1	0	1	1	0	1	0	1	1	0	0	0	1
ω_6	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0
ω_7	1	0	0	1	1	0	0	0	1	0	0	0	0	1	1
ω_8	1	0	1	1	1	0	1	1	0	0	0	0	1	0	0

Example: Error Correcting Output Codes, Majority Voting, and Decision Templates. Table 8.2 shows the code matrix for $c = 8$ classes and codeword length $L = 15$ found with the random generation method ($H_c = 5$, $H_L = 1$).

Suppose that the ensemble $\mathcal{D} = \{D_1, \ldots, D_{15}\}$ produces the following set of outcomes for some input $\mathbf{x}$

$$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] \tag{8.6}$$

The Hamming distances to the eight codewords are as follows: 8, 6, 9, 8, 6, 11, 9, 8. There is a tie between ω_2 and ω_5, so any of the two labels can be assigned.

By giving label $s_1 = 1$, classifier D_1 votes for classes ω_1, ω_2, ω_3, ω_5, ω_7, and ω_8.

The Hamming distance between binary vectors is equal to the Euclidean distance between them. If we regard the codewords as the templates for the classes, then by labeling $\mathbf{x}$ according to the minimal Hamming distance, we implement the decision template combiner.

8.2.4 Soft Error Correcting Output Code Labels and Pairwise Classification

Kong and Dietterich [261] use soft labels $d_{i,j}(\mathbf{x}) = \hat{P}(\Omega_i^{(j)}|\mathbf{x})$ instead of the $0/1$ labels from the L classifiers in the ensemble $j = 1, 2$. Since there are only two compound classes, $d_{i,2}(\mathbf{x}) = 1 - d_{i,1}(\mathbf{x})$. The Hamming distance between the classifier outputs and the codeword for class ω_k is

$$\mu_k(\mathbf{x}) = \sum_{i=1}^{L} |d_{i,1}(\mathbf{x}) - C(k, i)|, \qquad k = 1, \ldots, c \tag{8.7}$$

Note that only $d_{i,1}(\mathbf{x})$ is sufficient for the calculation. Suppose that in the eight-class example above the classifier outputs were

$$[0.2, 0.7, 0.3, 0.3, 0.9, 0.9, 0.5, 1.0, 0.9, 0.9, 0.4, 0.4, 0.6, 0.2, 0.0] \tag{8.8}$$

TABLE 8.3 Possible Scenarios in Voting if Classifier D_k Suggests Label ω_i in Solving a Pairwise Dichotomy Between ω_i and ω_j.

True label →	ω_i	ω_j	Another
Vote for ω_i	No error	Error	Error
Vote against ω_j	No error	Error	**No error**

The eight Hamming distances in this case are 6.8, 7.0, 5.6, 4.4, 6.6, 7.0, 9.2, 7.6 and the ensemble label for the input $\mathbf{x}$ is ω_4.

Pairwise discrimination between classes has been suggested in Refs. [257,262,263]. In this model there are $c \times (c-1)/2$ possible dichotomies, one for each pair of classes. The code word for class ω_j will contain "don't care" symbols to denote the classifiers that are not concerned with this class label. If we use all pairwise dichotomies, each code vector will have $c-1$ informative bits and $(c-1) \times (c-2)/2$ "don't care" bits. The class label for an input $\mathbf{x}$ is inferred again from the similarity between the codewords and the outputs of the classifiers. This method is impractical for a large c as the number of classifiers becomes prohibitive.

Cutzu suggests that we may wish to use only a subset of all possible pairwise classifiers. Then instead of the *for* votes, we should count the *against* votes [262]. There is a good intuitive reason for that. Let D_k be the classifier discriminating between ω_i or ω_j. Table 8.3 shows the possible scenarios in voting if D_k suggests ω_i. It is intuitively clear that the against vote is the safer option.

Cutzu explains that counting the against votes will be equivalent to counting the for votes if all pairwise classifiers are used. However, if only part of the possible pairwise classifiers are used as the ensemble, counting the against votes is the better method [262].

8.2.5 Comments and Further Directions

Kong and Dietterich study the reason behind the success of ECOC ensembles [261]. They use a decomposition of the error rate into bias and variance components and show that ECOC ensembles reduce both bias and variance. The variance is reduced by the very fact that multiple instable classifiers are used. Variance reduction is a characteristic of any homogeneous voting scheme where the same classifier model is applied multiple times. Bagging is one such algorithm as discussed in Chapter 7. Kong and Dietterich also state that bias is not reduced by such homogeneous voting schemes. They attribute the reduction in the bias in ECOC ensembles to the diversity in the errors of the classifiers. This diversity is due to the fact that different boundaries are being learned by the classifiers. They conjecture that classifiers with "global behavior" will benefit from ECOC (e.g., multilayer perceptrons and decision trees) whereas classifiers with "local behavior" will not benefit

from ECOC in such a degree (e.g., k-nearest neighbor, Parzen and RBF neural networks).

Computational complexity of ECOC ensembles has been identified in Ref. [261] as a problem for future research. The success of ECOC ensembles depends on the difference between the codewords. For a greater difference the codeword length L should be large. This means that a large number of decision trees or neural networks have to be stored and run for each new object submitted for classification. Such ECOC systems have been applied for cloud classification [254] and electronic noses [264].

8.3 COMBINING CLUSTERING RESULTS

Recall the clustering task in pattern recognition from Chapter 1. We are looking for groups (clusters) in the data. The members of one cluster should be similar to one another and dissimilar to the members of other clusters. A clustering algorithm produces a *partition* on an unlabeled data set $\mathbf{Z}$, $P = (\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(c)})$, such that no two clusters intersect and the union of all clusters is the data set itself.

8.3.1 Measuring Similarity Between Partitions

8.3.1.1 Mutual Information. The difficulty in finding a measure of similarity between partitions is that the labels are symbolic and any permutation of the symbols represents the same partition. This would be a problem if we want to match the labels directly. For example, the following two partitions of nine objects into three classes are identical but a direct comparison will result in no match at all.

Object	1	2	3	4	5	6	7	8	9
Partition 1	1	1	1	2	2	2	3	3	3
Partition 2	3	3	3	1	1	1	2	2	2

We can fix the labels of Partition 1, enumerate all the permutations of the labels of Partition 2 and pick the labeling that maximizes some direct measure of match. The simplest direct measure of match is the *proportion of matched labels*.

A way to avoid relabeling is to treat the two partitions as (nominal) random variables, say X and Y, and calculate the *mutual information* [265–267] between them. Consider the confusion matrix for partitions A and B where the rows correspond to the clusters in A and the columns correspond to the clusters in B. Denote by N_{ij} the (i, j)th entry in this confusion matrix, where N_{ij} is the number of objects in both cluster i of partition A and cluster j in partition B. Denote by $N_{i.}$ the sum of all columns for row i; thus $N_{i.}$ is the number of objects in cluster i of partition A. Define $N_{.j}$

to be the sum of all rows for column i, that is, $N_{.j}$ is the number of objects in cluster j in partition B. (There is no requirement that the number of clusters in A and B should be the same.) Let c_A be the number of clusters in A and c_B be the number of clusters in B. The mutual information between partitions A and B is[33]

$$MI(A, B) = \sum_{i=1}^{c_A} \sum_{j=1}^{c_B} \frac{N_{ij}}{N} \log\left(\frac{N_{ij}N}{N_{i.}N_{.j}}\right) \tag{8.9}$$

A measure of similarity between partitions A and B is the *normalized* mutual information [265][34]

$$NMI(A, B) = \frac{-2\sum_{i=1}^{c_A}\sum_{j=1}^{c_B} N_{ij} \log\left(\frac{N_{ij}N}{N_{i.}N_{.j}}\right)}{\sum_{i=1}^{c_A} N_{i.} \log\left(\frac{N_{i.}}{N}\right) + \sum_{j=1}^{c_B} N_{.j} \log\left(\frac{N_{.j}}{N}\right)} \tag{8.11}$$

If A and B are identical, then NMI takes its maximum value of 1. If A and B are independent, that is, having complete knowledge of partition A, we still know nothing about partition B and vice versa, then

$$\frac{N_{ij}N}{N_{i.}N_{.j}} \to 1, \qquad \text{for any } i, j, \qquad \text{and} \qquad NMI(A, B) \to 0$$

Example: Normalized Mutual Information (*NMI*) Measure for Comparing Partitions. Consider partitions A and B with the following labeling

Object	1	2	3	4	5	6	7	8
Partition A	1	1	2	2	3	3	4	4
Partition B	2	1	3	3	2	1	2	2

[33] By convention, $0 \times \log(0) = 0$.

[34] The *NMI* proposed by Strehl and Ghosh [266] is

$$NMI(A, B) = \frac{\sum_{i=1}^{c_A}\sum_{j=1}^{c_B} N_{ij} \log\left(\frac{N_{ij}N}{N_{i.}N_{.j}}\right)}{\sqrt{\sum_{i=1}^{c_A} N_{i.} \log\left(\frac{N_{i.}}{N}\right) \sum_{j=1}^{c_B} N_{.j} \log\left(\frac{N_{.j}}{N}\right)}} \tag{8.10}$$

The confusion matrix between the two partitions is

	B →			
A ↓	1	2	3	Total
1	1	1	0	2
2	0	0	2	2
3	1	1	0	2
4	0	2	0	2
Total	2	4	2	

Using Eq. (8.10),

$$NMI(A, B) = \frac{-2 \times 5.5452}{-11.0904 - 8.3178} \approx 0.5714 \tag{8.12}$$

The value of *NMI* will be the same for any permutation of the labels of A or B.

8.3.1.2 Rand Index. Rand [268] proposes a simple measure of agreement between two partitions A and B. Denote by n_{11} the number of *pairs* of objects from $\mathbf{Z}$, which are both in the same cluster in A and are also both in the same cluster in B. Let n_{00} be the number of *pairs* of objects from $\mathbf{Z}$ that are in different clusters in A and are also in different clusters in B. Both n_{00} and n_{11} are agreement quantities as in both partitions the pair of objects have been found to be either similar enough so as to be placed in the same cluster or dissimilar enough so as to be placed in different clusters. Accordingly, we can define the two disagreement quantities n_{01} and n_{10}. Note that there are $N(N-1)/2$ possible pairs of objects in $\mathbf{Z}$, therefore

$$n_{00} + n_{11} + n_{01} + n_{10} = \frac{N(N-1)}{2} \tag{8.13}$$

The Rand index is

$$r(A, B) = \frac{n_{00} + n_{11}}{n_{00} + n_{11} + n_{01} + n_{10}} = \frac{2 * (n_{00} + n_{11})}{N(N-1)} \tag{8.14}$$

The Rand index takes value 1 if the partitions agree completely (regardless of the permutation of the labels), but does not have a constant value for the case when both partitions are drawn at random.

8.3.1.3 Jaccard Index. Using the same notation as for the Rand index, the *Jaccard index* between partitions A and B is [269]

$$J(A, B) = \frac{n_{11}}{n_{11} + n_{01} + n_{10}} \tag{8.15}$$

8.3.1.4 Adjusted Rand Index. The *adjusted Rand index* corrects for the lack of a constant value of the Rand index when the partitions are selected at random [270]. Suppose that the two partitions A and B are drawn randomly with a fixed number of clusters and a fixed number of objects in each cluster (generalized hypergeometric distribution). The expected value of the adjusted Rand index for this case is zero. The adjusted Rand index, ar, is calculated from the values N_{ij} of the confusion matrix for the two partitions as follows

$$t_1 = \sum_{i=1}^{c_A} \binom{N_{i.}}{2}; \qquad t_2 = \sum_{j=1}^{c_B} \binom{N_{.j}}{2}; \tag{8.16}$$

$$t_3 = \frac{2t_1 t_2}{N(N-1)}; \tag{8.17}$$

$$ar(A, B) = \frac{\sum_{i=1}^{c_A} \sum_{j=1}^{c_B} \binom{N_{ij}}{2} - t_3}{\frac{1}{2}(t_1 + t_2) - t_3} \tag{8.18}$$

where $\binom{a}{b}$ is the binomial coefficient $a!/b!(a-b)!$. The value of the adjusted Rand index for partitions A and B from the previous example is

$$ar(A, B) = \frac{2 - \frac{8}{7}}{\frac{1}{2}(4+8) - \frac{8}{7}} = \frac{3}{17} \tag{8.19}$$

8.3.2 Evaluating Clustering Algorithms

Rand [267] proposes four scenarios for evaluating a clustering algorithm:

1. *Retrieval*. The principal problem in clustering is that there are no preassigned labels on the data set. However, we can generate artificial data with a desired cluster structure and use this structure as a ground truth. Typical examples are clouds of points generated from a multidimensional normal distribution; geometrical figures such as our banana examples; and uniformly distributed points.
2. *Sensitivity to perturbation*. A reliable clustering algorithm should not be affected too much by small perturbations of the data set. We can test this property by adding a small amount of noise to each point and compare the resultant clustering with the clustering of the noise-free data [268]. Another option is to

compare pairs of partitions obtained by injecting noise and draw conclusions on the basis of the distribution of the pairwise similarities.

3. *Sensitivity to missing values*. Clustering should not be affected to a large extent if a small part of the data set is missing. To evaluate a clustering algorithm, we can subsample from $\mathbf{Z}$ (take $M < N$ objects without replacement) and run the algorithm on each sample. The similarity between each pair of clustering results is calculated and the distribution of these similarities is used to judge how sensitive the clustering algorithm is. An algorithm for obtaining a set of pairwise similarities, called "the model explorer algorithm" is proposed by Ben-Hur et al. [269] (Figure 8.3). They apply this model for different values of c (the number of clusters), and estimate the most likely number of clusters in data by inspecting the histograms of the distributions of pairwise similarities. The details of the model-explorer procedure are explained in an example below.
4. *Comparison to other clustering algorithms*. Clustering algorithms are usually devised to optimize different criterion functions. These functions are chosen to represent the concept of a "good clustering." For example, a "good cluster" might be defined as a *compact* one, that is, a cluster where the distances between the points and the cluster centroid are small. Another equally intui-

Model explorer algorithm

(for a fixed number of clusters, c)

1. Pick the number of clusters c, the number of pairs of subsamples L, and a similarity measure $S(A, B)$ between two *partitions* A and B. Specify the proportion of objects, f, to be sampled from $\mathbf{Z}$ without replacement. (Recommended value is $f = 0.8$.)
2. Generate two subsamples from $\mathbf{Z}$, S_1 and S_2, of size fN each.
3. Cluster S_1 into c clusters; denote this partition by $A(S_1)$.
4. Cluster S_2 into c clusters; denote this partition by $B(S_2)$.
5. Find the objects present in both subsamples, i.e., $S_{12} = S_1 \cap S_2$.
6. Calculate and store the similarity between the restricted partitions $\mathcal{S}(A(S_{12}), B(S_{12}))$.
7. Repeat steps 2 to 6 L times.

Fig. 8.3 *The model explorer algorithm of Ben-Hur et al. [269] for a fixed value of the number of clusters, c.*

tive definition of a good cluster is that each point shares the same cluster label as its nearest neighbor. The algorithms implementing these two definitions might lead to very different partitions of the same data set. Comparison of clustering algorithms that optimize different criteria only makes sense when we have a ground truth label or when the goodness of the clusterings can be evaluated visually (the retrieval task).

Example: The Model Explorer Algorithm for the Banana Data. The model explorer algorithm was run with the single linkage clustering, for $c = 2, 3, \ldots, 10$ clusters and $L = 200$ pairs of subsamples, with the banana data in Figure 1.16. The presumption is that if there is a structure in the data, it will be found even if small changes of the data set are made, for example, removing, adding, or slightly offsetting a small fraction of the points [269]. Therefore, the partitions of the perturbed data sets will be similar to one another. Conversely, if there is no structure in the data, then partitions will be arbitrary and will exhibit larger spread of the pairwise similarity. Ben-Hur et al. [269] suggest to inspect the histograms of the similarity and choose the largest number of clusters, c, for which the similarity has consistently high values. Figure 8.4 shows the results for the banana data using the Jaccard index (8.15) as the similarity measure.

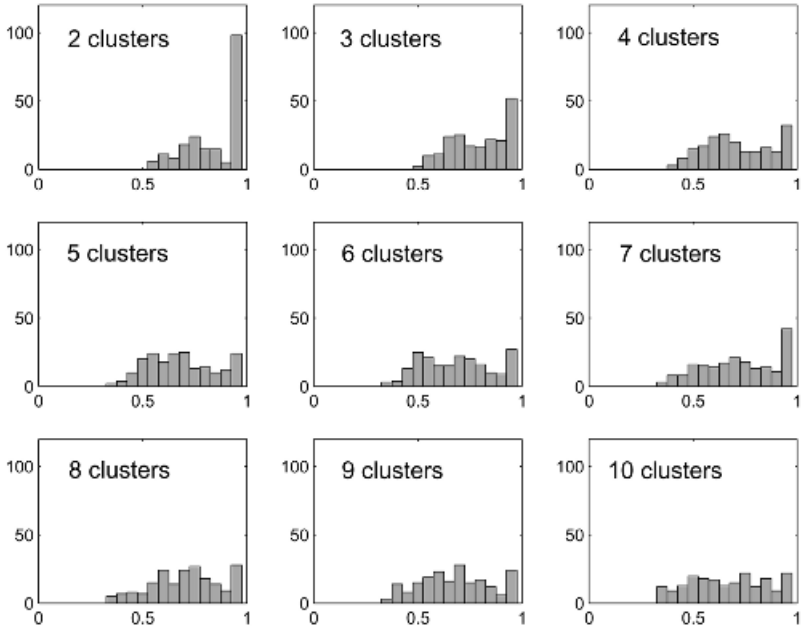


Fig. 8.4 *Histograms of the Jaccard index of similarity between pairs of partitions randomly drawn from the banana data according to the model-explorer algorithm* ($L = 200$, $f = 0.8$). *The single linkage algorithm was used to obtain the partitions.*

The histograms suggest that the data set is most likely to have two clusters. A piece of Matlab code for the model explorer algorithm is given in Appendix 8C. The single linkage clustering procedure can be replaced by any other procedure that produces a vector of numerical labels taking values $1, 2, \ldots, c$. The function for calculating the Jaccard index is also given.

8.3.3 Cluster Ensembles

Combining the results of several clustering methods has recently resurfaced as one of the branches of multiple classifier systems [122,265–267,271–275]. The aim of combining partitions is to improve the quality and robustness of the results. Choosing a single clustering algorithm for the problem at hand requires both expertise and insight, and this choice might be crucial for the success of the whole study. Selecting a clustering algorithm is more difficult than selecting a classifier. In classification we may run cross-validation pilot experiments and determine which model to use. For clustering, no ground truth is available and the results are not easy to assess. Therefore, instead of running the risk of picking an unsuitable clustering algorithm, a *cluster ensemble* can be used [266].

Based on potential applications in data mining, distributed computing, finance data analysis, gene expression analysis, and so on, Strehl and Ghosh identify two major uses of cluster ensembles [266]:

1. *Knowledge reuse*. In some application domains, various clusterings of the data have already been made, either by data analysis or by predefined grouping rules. Each partition is a piece of knowledge about the object. We aim at re-using this knowledge to build up a single resultant clustering.
2. *Distributed clustering*. Two subdomains of distributed clustering are *feature-distributed* clustering and *object-distributed clustering*. Feature distributed clustering means that, for some reason, the whole data set cannot be processed on one machine or by one clustering procedure. The reason could be computational or one of privacy and security of the raw data. L partitions of the data set are built separately, using different subsets of features, and then aggregated into a single partition. In object-distributed clustering the objects are clustered separately. A clustering will therefore be performed on the regional data and the characteristics of the clusters will be aggregated at the combiner level.

In classifier combination *diversity* is vital for the success of the ensemble. To ensure that the cluster ensemble is diverse we can use the same strategies as for building classifier ensembles:

- *Different subsets of features.* Each partition is found using only a subset of the original feature set. The subsets of features might be overlapping or disjoint.
- *Different clustering algorithms.* We can combine hierarchical and nonhierarchical algorithms, sequential or batch algorithms, random or deterministic algorithms, and so on.

- *Randomizing*. Some clustering algorithms are inherently random, for example, those based on genetic algorithms or simulated annealing ideas. Thus the ensemble can be formed as the result of L different runs of the algorithm. Other clustering algorithms, such as c-means, only need a random initialization and then proceed in a deterministic way. To compose a cluster ensemble, we can run such an algorithm from L different initial starting points. Algorithms that use the data in a sequential way, for example, vector quantization, may produce different results if the data set is reordered before feeding it to the algorithm.
- *Different data sets.* Resampling with or without replacement is one possible route for building a cluster ensemble [265,275].

8.3.3.1 Majority Vote Clustering. Figure 8.5 shows a generic *cluster ensemble* algorithm. For a given data set $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$, L ensemble members are generated by clustering $\mathbf{Z}$ or a subsample of it. For each clustering, a *co-association matrix* of size $N \times N$ is formed [265,271–273], denoted $M^{(k)}$, termed sometimes connectivity matrix [275] or similarity matrix [266]. Its (i, j)th entry is 1 if $\mathbf{z}_i$ and $\mathbf{z}_j$ are in the same cluster in partition k, and 0, otherwise. A final matrix $\mathbf{M}$ is derived from $M^{(k)}$, $k = 1, \ldots, L$, called a *consensus matrix* in Ref. [275]. The final clustering is decided using $\mathbf{M}$. The number of clusters may be prespecified or found through further analysis of $\mathbf{M}$.

Perhaps the simplest implementation of the generic cluster ensemble algorithm is as follows (called "voting c-means algorithm" in Ref. [272] and "evidence accumulation algorithm" in Ref. [273]).

Cluster ensemble algorithm

1. Given is a data set $\mathbf{Z}$ with N elements. Pick the ensemble size L and the number of clusters c.
2. Generate L partitions of $\mathbf{Z}$ in c clusters.
3. Form a co-association matrix for each partition, $M^{(k)} = \left\{m_{ij}^{(k)}\right\}$, of size $N \times N$, $k = 1, \ldots, L$, where

$$m_{ij}^{(k)} = \begin{cases} 1, & \text{if } \mathbf{z}_i \text{ and } \mathbf{z}_j \text{ are in the same cluster in partition } k, \\ 0, & \text{if } \mathbf{z}_i \text{ and } \mathbf{z}_j \text{ are in different clusters in partition } k \end{cases}$$

4. Form a final co-association matrix $\mathbf{M}$ (consensus matrix) from $M^{(k)}$, $k = 1, \ldots, L$, and derive the final clustering using this matrix.

Fig. 8.5 *A generic cluster ensemble algorithm.*

1. Pick the initial number of clusters c, the ensemble size L, and a threshold θ, $0 < \theta < 1$.
2. Run c-means L times, with c clusters, and form $M^{(1)}, \ldots, M^{(L)}$.
3. Calculate $\mathbf{M} = \frac{1}{L}(M^{(1)} + \cdots + M^{(L)})$.
4. "Cut" $\mathbf{M}$ at threshold θ. Join in the same cluster all the points whose pairwise entry in $\mathbf{M}$ is greater than θ. For all the remaining points form single-element clusters.

This implementation is based on the majority vote. If points $\mathbf{z}_i$ and $\mathbf{z}_j$ have been in the same cluster in the majority of the L partitions, then they will be assigned to the same cluster in the final partition. The final number of clusters is not prespecified; it depends on the threshold θ and on the number of initial clusters c. The combination of these two parameters is crucial for discovering the structure of the data. The rule of thumb is $c = \sqrt{N}$. Fred and Jain [273] also consider fitting a mixture of Gaussians to $\mathbf{Z}$ and taking the identified number of components as c. Neither of the two heuristics works in all the cases. Fred and Jain conclude that the algorithm can find clusters of any shape but it is not very successful if the clusters are touching. Figure 8.6 shows the results from applying the voting c-means algorithm to the banana data.

The consensus matrix $\mathbf{M}$ can be regarded as a similarity matrix between the points on $\mathbf{Z}$. Therefore, it can be used with any clustering algorithm that operates directly upon a similarity matrix. In fact, "cutting" $\mathbf{M}$ at a certain threshold is equivalent to running the single linkage algorithm and cutting the dendrogram obtained from the hierarchical clustering at similarity θ. Viewed in this context, the cluster ensemble is a type of *stacked clustering* whereby we can generate layers of similarity matrices and apply clustering algorithms on them. Figure 8.7 shows the results of the stacked clustering for the banana data. We ran single linkage clustering on $\mathbf{M}$ and stopped at two clusters. The misclassified objects are encircled.

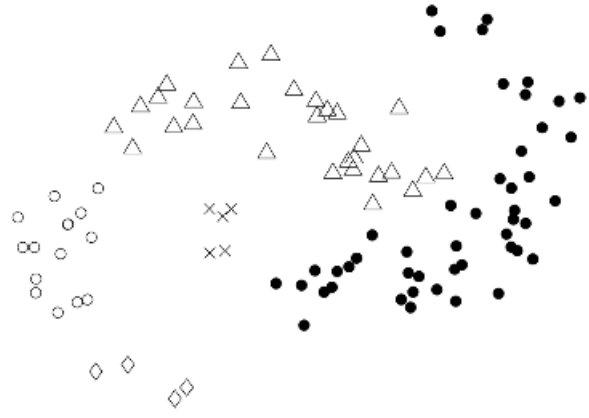


Fig. 8.6 *Majority vote clustering: Results from the combination of $L = 50$ c-means clusterings from different initializations ($c = 10$, $\theta = 0.5$).*

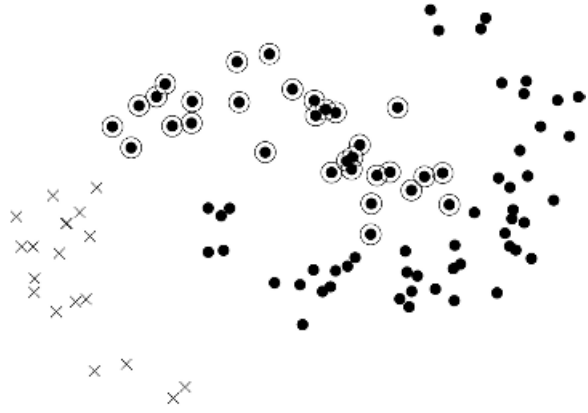


Fig. 8.7 *Stacked clustering with two target clusters: Results from the combination of* $L = 50$ *c-means clusterings from different initializations* ($c = 10$). The misclassified objects are encircled.

To measure the match between the true banana labels and the obtained labels, we used the Rand index, Jaccard index, the adjusted Rand index, and the normalized mutual information index (*NMI*). Table 8.4 shows the values of the indices for the two individual clusterings (single linkage and *c*-means), the majority vote clustering, and the stacked clustering. Majority vote scores higher than the rest of the methods because the two bananas can be assembled from the identified clusters with virtually no error and because of the higher number of final clusters. The remaining three clustering methods are rated differently by the similarity indices, which is yet another hint for the difficulties in judging clustering results.

It should be noted that the number of parameters that need to be specified in the majority vote clustering and the stacked clustering is the same. Instead of a threshold θ, in stacked clustering we have to specify the desired number of clusters.

A suitable number of clusters can be derived from the single linkage clustering using $\mathbf{M}$ as the similarity matrix [265,272,273]. At step i, hierarchical clustering merges clusters at distance d_i (note that d_i increases with i). Suppose that the distance between the clusters is associated with some energy needed to join the clusters. If the distance is small, then a small amount of energy is needed and joining is easy.

TABLE 8.4 Similarity Indices Between the Original Banana Labels and the Labels from Four Clustering Methods.

Clustering Method	Rand	Jaccard	Adjusted Rand	*NMI*
Single linkage	0.4982	0.4771	0.0049	0.0663
c-means	0.5927	0.4289	0.1858	0.1587
Majority vote	0.7935	0.5976	0.5856	0.6197
Stacked clustering	0.5679	0.4653	0.1390	0.2615

Define $\Delta_i = d_i - d_{i-1}$ to be the *additional* energy from step $i-1$ to step i for the merge to occur. If this energy is too high, then this might mean that we are forcing the system into a structure that it does not have. Therefore we should stop the process at the highest Δ_i and return the $N - i + 1$ clusters at step $i - 1$. In other words, we should cut the dendrogram resulting from the hierarchical clustering at the largest jump of the distance criterion.

Fred and Jain propose a framework for analyzing majority vote clustering ensembles [265]. They use the normalized mutual information criterion (*NMI*) to measure the consistency between the partition produced by the ensemble and the L individual partitions. Let $\mathcal{P} = \{P^1, \ldots, P^L\}$ be the ensemble and let P^* be the resultant partition. The consistency of the final partition with the ensemble is measured as

$$NMI(P^*, \mathcal{P}) = \frac{1}{L} \sum_{i=1}^{L} NMI(P^*, P^i) \tag{8.20}$$

Let c be the supposed number of clusters in $\mathbf{Z}$. The optimal resultant partition in c clusters will be the one that maximizes $NMI(P^*, \mathcal{P})$. To select the optimal number of clusters c, Fred and Jain [265] propose to study the variance of $NMI(P^*, \mathcal{P})$ computed across small perturbations in $\mathcal{P}$. The value of c should be the one corresponding to the lowest variance. The perturbations are generated by taking bootstrap samples of the *partitions* in $\mathcal{P}$. Suppose that $\mathcal{P}^b$ is an ensemble formed by a bootstrap sample from the partitions in $\mathcal{P}$. The ensemble partition, P^{*b}, is calculated and its consistency with the ensemble is measured by $NMI(P^{*b}, \mathcal{P}^b)$. This procedure is repeated B times for each c. The variance of $NMI(P^{*b}, \mathcal{P}^b)$ across all B bootstrap ensembles is taken to be a measure of the robustness of the partition for the respective c.

Monti et al. [275] suggest using resampling from $\mathbf{Z}$, either with or without replacement. Data sets $S_1, \ldots, S_L$ are generated through resampling and each is clustered using a chosen clustering algorithm. Since not all objects will be included in all the data sets, the co-association matrices will not be completely filled. A co-association matrix $M^{(i)}$ will have entries 1 and 0 for the elements of $\mathbf{Z}$ that are in S_i and "blanks" for the remaining elements of $\mathbf{Z}$. The overall consensus matrix $\mathbf{M}$ will be again the average of the L individual co-association matrices ignoring the blank entries. For example, suppose that entries $m_{2,7}$ in $M^{(1)}, \ldots, M^{(10)}$ are [0, 0, 1, #, #, 1, 1, 0, #, 1], respectively, where # denotes a blank entry. Then $\mathbf{m}_{2,7} = 4/7$. Monti et al. propose to run the clustering for the individual ensemble members for a range of c and analyze $\mathbf{M}$ in order to select the best c. If there was a clear-cut structure in the data set, then all the individual algorithms would identify it, and the consensus between them would be high. This is shown by values close to 1 and 0 in $\mathbf{M}$. Hence the distribution of the values in $\mathbf{M}$ can be an indication of the confidence of the ensemble about the currently used value of c. The optimal number of clusters is decided as the one for which $\mathbf{M}$ shows the highest confidence. The consensus matrix $\mathbf{M}$ is then submitted to a clustering algorithm together with the optimal c (stacked clustering as discussed earlier) to derive the final partition.

The generic cluster ensemble method requires a co-association matrix of size $N \times N$, which might be infeasible for large data sets [276]. To reduce the computational complexity Fred and Jain propose to reduce the $N \times N$ co-association matrix to an $N \times p$ matrix where $p < N$. Only the p nearest neighbors of each element in $\mathbf{Z}$ participate in the co-association matrices throughout the algorithm.

8.3.3.2 Direct Optimization in Cluster Ensembles. If we had a reliable way to solve the correspondence problem between the partitions, then the voting between the clusterers would be straightforward: just count the number of votes for the respective cluster. The problem is that there are $c!$ permutations of the labels and an exhaustive experiment might not be feasible for large c. However, for small c we can still be able to solve the correspondence problem and run a direct majority vote clustering.

Let $d_{i,j}(\mathbf{x})$ be the membership to cluster j in partition P^i. Denote by $\mu_k(\mathbf{x})$ the membership to cluster k in partition P^* found as the ensemble output. Weingessel et al. [276] propose a sequential cluster ensemble algorithm that goes through the following steps:

1. Choose the number of iterations (ensemble members) L and the number of clusters c.
2. Run a clustering algorithm to find the first partition P^1. Initialize the ensemble partition P^* as $P^* = P^1$. Set $i = 2$.
3. Run a clustering algorithm to find partition P^i.
4. Permute the labels of P^i to find the labeling that maximizes the similarity between P^* and P^i. (Similarity is defined as the trace of the confusion matrix between the two partitions.)
5. Using the optimal labeling of P^i, update the membership of object $\mathbf{z}_j \in \mathbf{Z}$ in cluster k at step i as

$$\mu_k^{\{i\}}(\mathbf{z}_j) = \frac{i-1}{i}\mu_k^{\{i-1\}}(\mathbf{z}_j) + \frac{1}{i}d_{i,k}(\mathbf{z}_j), \qquad j = 1, \ldots, N \tag{8.21}$$

   where the superscript of μ denotes the iteration number and $k = 1, \ldots, c$.
6. Increment i and repeat steps 3 to 5 until $i > L$. Return $P^* = P^L$.

Finally $\mathbf{z}_j$ is assigned to the cluster with the largest $\mu_k(\mathbf{z}_j)$ in the final partition, that is,

$$\arg\max_k \mu_k^{\{L\}}(\mathbf{z}_j), \qquad j = 1, \ldots, N \tag{8.22}$$

This algorithm can be used with fuzzy clustering as the cluster membership $d_{i,k}(\mathbf{z}_j)$ is not restricted to zero or one.

If c is large, the correspondence problem between partitions A and B can be solved approximately by the following greedy algorithm [276]:

1. Find the confusion matrix between A and B.
2. Find the maximum element in the matrix.
3. Relate the two respective clusters and drop them from both partitions.
4. Reduce the confusion matrix and repeat steps 2 and 3 until all correspondences are resolved.

The Operational Research field offers a solution to the assignment problem by the so called Hungarian algorithm [277].

Strehl and Ghosh [266] consider a direct optimization method for finding the resultant partition, P^*, for a given ensemble. The objective is to maximize *NMI* between P^* and the ensemble, that is, find

$$NMI(P^*, \mathcal{P}) = \max_P NMI(P, \mathcal{P}) \tag{8.23}$$

where P varies among the possible

$$\frac{1}{c!}\sum_{i=1}^{c}\binom{c}{i}(-1)^{(c-i)}i^N \tag{8.24}$$

partitions of N objects into c clusters [37]. To avoid the task of enumerating all partitions and calculating the *NMI* for each one, Strehl and Ghosh devise a greedy algorithm that sequentially relabels the points of $\mathbf{Z}$ until a local maximum of *NMI* is reached. The algorithm operates as follows:

1. Choose partition P^k among $P^1, \ldots, P^L$, which has the largest mutual information with the ensemble, that is,

$$NMI(P^k, \mathcal{P}) = \max_{i=1}^{L} NMI(P^i, \mathcal{P}) \tag{8.25}$$

   to be the current partition.
2. Start a "sweep" through $\mathbf{Z}$: For each $\mathbf{z}_j \in \mathbf{Z}$ change its label in the current partition to any of the other $c-1$ labels. Select the label that maximizes *NMI* between the current partition and the ensemble.
3. If a label change has occurred during the sweep, arrange the data set $\mathbf{Z}$ in a random order and start another sweep; else STOP and return the current partition.

Besides being nonoptimal, this algorithm has been found to be very slow [266]. Strehl and Ghosh propose three alternative heuristic algorithms that are less time consuming. Since all these algorithms are optimizing the same criterion (*NMI*), they can be included in a meta-scheme whereby all three are tried on $\mathbf{Z}$ and the best algorithm is chosen.

The ultimate goal of cluster ensemble methods is to provide a robust clustering tool that does not rely on great expertise on the user's part or on a lucky guess. The authors of the methods and algorithms presented in this section have given experimental verification of the methods' potential for that with both artificial and real data.

APPENDIX 8A EXHAUSTIVE GENERATION OF ERROR CORRECTING OUTPUT CODES

The code below generates a code matrix C with an exhaustive $ECOC$ for c number of classes.

Since there are exactly $2^{(c-1)} - 1$ different assignments, the exhaustive code will be obtained if we enumerate the numbers from 0 to $2^{(c-1)} - 2$, convert them from decimal to binary and append a string of 1s to be the first row of the code matrix. The code line below produces a code matrix of string type.

```
C=[num2str(ones(2^(c-1)-1,1))';dec2bin(0:2^(c-1)-2)']
```

APPENDIX 8B RANDOM GENERATION OF ERROR CORRECTING OUTPUT CODES

The Matlab code below was used to produce the example for $c = 8$ classes and $L = 15$ classifiers. The function `ecoc_hamming` takes as its input a $c \times L$ code matrix P and outputs the minimum row (class) Hamming distance D and minimum column (classifier) Hamming distance DD. The resultant code matrix is stored in PP with minimum row Hamming distance H_c and minimum column Hamming distance H_L.

```
c=8;L=15;
Hc=0;HL=0;T=500;
for t=1:T,
  P=rand(c,L)>0.5;
  [D,DD]=ecoc_hamming(P);
  if D+DD>Hc+HL
    PP=P; % best code matrix
    Hc=D;
    HL=DD;
  end
end

function [D,DD]=ecoc_hamming(P);
[c,L]=size(P);
D=L;  % Find the two closest rows (codewords)
for j=1:c-1
```

```
   for k=j+1:c
       D=min(D,sum(abs(P(j,:)-P(k,:))));
   end;
end;
DD=c;  % Find the two closest columns (classifiers)
for j=1:L-1
   for k=j+1:L
       DD=min(DD,min(sum(abs(P(:,j)-P(:,k))),...
                     sum(abs(1-P(:,j)-P(:,k)))));
   end;
end;
```

APPENDIX 8C MODEL EXPLORER ALGORITHM FOR DETERMINING THE NUMBER OF CLUSTERS *C*

The code below produces an array r with L rows and nine columns (for number of clusters $c = 2, \ldots, 10$) containing the Jaccard index of similarity between partitions (8.15). The (i,j)th entry in r is the Jaccard index for the ith pair of partitions, with $j+1$ clusters in each. The function `single_linkage(Z,c)` can be replaced by any other clustering algorithm that takes a data set $\mathbf{Z}$ and the desired number of clusters, c, and produces a vector with class labels for the elements of $\mathbf{Z}$.

```
L=200;f=0.8;N=size(Z,1);
template=zeros(N,1);
for classes=2:10,
   for i=1:L,
     S1=rand(N,1)<f;
     S2=rand(N,1)<f;
     S=S1&S2;
     A=single_linkage(Z(find(S1),:),classes);
     labelA=template;labelA(find(S1))=A;
     B=single_linkage(Z(find(S2),:),classes);
     labelB=template;labelB(find(S2))=B;
     r(i,classes-1)=Jaccard_index(labelA(find(S)),...
                                  labelB(find(S)));
   end;
end;
```

The function `Jaccard_index` calculates the Jaccard coefficient for two label vectors A and B. The same code can be used for the Rand index by adding at the end `r=agreement/total_pairs;`.

```
function J=Jaccard_index(A,B);
N=length(A);
agreement=0;Jaccard=0;
for i=1:N-1,
   for j=i+1:N,
```

```
        logicalA=A(i)==A(j);
        logicalB=B(i)==B(j);
        if (logicalA&logicalB),
              Jaccard=Jaccard+1;
        end;
        if (logicalA&logicalB)|((~logicalA)&(~logicalB)),
              agreement=agreement+1;
        end;
    end;
end;
total_pairs=N*(N-1)/2;
J=Jaccard/(total_pairs-agreement+Jaccard);
```

9

Theoretical Views and Results

There is no general, agreed upon, underlying theory of classifier combination. There are, however, various results and ideas scattered in the literature. Some of these are nearly obvious but have not been formulated explicitly hitherto, for example, the equivalence of minimum and maximum combination rules for two classes (Section 9.1.1). Others require straightforward but relatively long algebraic manipulations, so only selected special cases are brought in. For example, we follow the derivation of a relationship between the individual and the ensemble errors only for unbiased classifiers (Section 9.2) and refer the reader to the relevant literature for further exploration of this line.

9.1 EQUIVALENCE OF SIMPLE COMBINATION RULES

9.1.1 Equivalence of MINIMUM and MAXIMUM Combiners for Two Classes

Let $\mathcal{D} = \{D_1, \ldots, D_L\}$ be the classifier ensemble and $\Omega = \{\omega_1, \omega_2\}$ be the set of class labels. The individual outputs are estimates of the posterior probabilities, that is, the output $d_{i,j}(\mathbf{x})$ of classifier D_i in support of the hypothesis that $\mathbf{x}$ comes from class ω_j is an estimate of $P(\omega_j|\mathbf{x})$, $j = 1, 2$. Here we prove that the minimum and the maximum combiners are equivalent for $c = 2$ classes and any number of classifiers L, provided the two outputs from each classifier satisfy

$$\hat{P}(\omega_1|\mathbf{x}) + \hat{P}(\omega_2|\mathbf{x}) = 1$$

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

This equivalence means that the class label assigned by one of the rules will be the same as the class label assigned by the other rule. In case of a tie for one of the rules, there will be a tie for the other rule as well, and any of the two class labels could be assigned in both cases.

Proposition. Let $a_1, \ldots, a_L$ be the L outputs for class ω_1, and $1 - a_1, \ldots, 1 - a_L$ be the L outputs for class ω_2, $a_i \in [0, 1]$. Then the class label assigned to $\mathbf{x}$ by the MINIMUM and MAXIMUM combination rules is the same.

Proof. Without loss of generality assume that $a_1 = \min_i a_i$, and $a_L = \max_i a_i$. Then the minimum combination rule will pick a_1 and $1 - a_L$ as the support for ω_1 and ω_2, respectively, and the maximum rule will pick a_L and $1 - a_1$. Consider the three possible relationships between a_1 and $1 - a_L$.

(a) If $a_1 > 1 - a_L$ then $a_L > 1 - a_1$, and the selected class is ω_1 *with both methods*;

(b) If $a_1 < 1 - a_L$ then $a_L < 1 - a_1$, and the selected class is ω_2 *with both methods*;

(c) If $a_1 = 1 - a_L$ then $a_L = 1 - a_1$, and we will pick a class at random *with both methods*.

Note, a discrepancy between the error rates of the two combination methods might occur in numerical experiments due to the random tie break in (c). If we agree to always assign class ω_1 when the support for the two classes is the same (a perfectly justifiable choice), the results for the two methods will coincide.

9.1.2 Equivalence of MAJORITY VOTE and MEDIAN Combiners for Two Classes and Odd Number of Classifiers

Consider again the case of two classes, and L classifiers with outputs for a certain $\mathbf{x}$, $a_1, \ldots, a_L$, for class ω_1, and $1 - a_1, \ldots, 1 - a_L$, for class ω_2, where L is odd.

Proposition. The class label assigned to $\mathbf{x}$ by the MAJORITY VOTE rule and MEDIAN combination rule is the same.

Proof. Again assume that $a_1 = \min_i a_i$, and $a_L = \max_i a_i$. Consider the median rule first. The median of the outputs for class ω_1 is $a_{L+1/2}$.

(a) If $a_{(L+1)/2} > 0.5$, then the median of the outputs for ω_2, $1 - a_{(L+1)/2} < 0.5$, and class ω_1 will be assigned. The fact that $a_{(L+1)/2} > 0.5$ means that all $a_{(L+1)/2+1}, \ldots, a_L$ are greater than 0.5. This makes at least $(L + 1)/2$ posterior probabilities for ω_1 greater than 0.5, which, when "hardened," will give label ω_1. Then the majority vote rule will assign to $\mathbf{x}$ class label ω_1.

(b) Alternatively, if $a_{(L+1)/2} < 0.5$, then $1 - a_{(L+1)/2} > 0.5$, and class ω_2 will be assigned by the median rule. In this case, at least $(L+1)/2$ posterior probabilities for ω_2 are greater than 0.5, and the majority vote rule will assign label ω_2 as well.

(c) For $a_{(L+1)/2} = 0.5$ a tie occurs, and any of the two labels can be assigned by the median rule. The same applies for the majority vote, as all the "soft" votes at 0.5 (same for both classes) can be "hardened" to any of the two class labels. Thus the majority can pull either way.

Again, a difference in the estimated errors of the two methods might occur in experiments due to the arbitrary "hardening" of label 0.5. For example, if we agree to always assign class ω_1 when the posterior probabilities are both 0.5, the results for the two methods will coincide.

9.2 ADDED ERROR FOR THE MEAN COMBINATION RULE

9.2.1 Added Error of an Individual Classifier

In a series of studies, Tumer and Ghosh analyze the theoretical improvement on the individual accuracy using different combination methods [278–280]. In this section we give their framework and follow their derivation for the added error of the classifier ensemble.

To make the analysis possible, we confine the study to the simple case of $x \in \Re$ instead of $\mathbf{x} \in \Re^n$. Two posterior probability functions $P(\omega_i|x)$ and $P(\omega_j|x)$ are depicted in Figure 9.1. Their intersection defines the optimal classification boundary x^*. Using x^* for labeling x (ω_i for $x \leq x^*$, ω_j for $x > x^*$), the Bayes error will correspond to the light gray area. Shown in Figure 9.1 by dashed lines are the (imperfect) approximations of the two functions by a hypothetical classifier. The approximations intersect at a different point, introducing an inaccurate classification boundary, denoted x_b. The dark gray area corresponds to the additional error incurred by using x_b instead of x^* for labeling x.

We assume that at each x, the approximations can be expressed as

$$\hat{P}(\omega_i|x) = P(\omega_i|x) + \varepsilon_i(x) \tag{9.1}$$

where $P(\omega_i|x)$ is the true posterior probability and $\varepsilon_i(x)$ is an error term.

Note that the exact shape of the approximations is irrelevant for the added error. The important requirement is that wherever the true probability for class ω_i dominates the probabilities for the other classes, the approximation for this class also dominates the other approximations. In the example in Figure 9.1, $P(\omega_j|x)$ is the dominating probability in the interval $[x^*, x_b]$ but the approximation erroneously places $P(\omega_i|x)$ as the higher valued function in this interval.

Next we evaluate the effect of the offset $b = x_b - x^*$ on the accuracy of the hypothetical classifier. At the new boundary, $x_b = x^* + b$, the approximations are

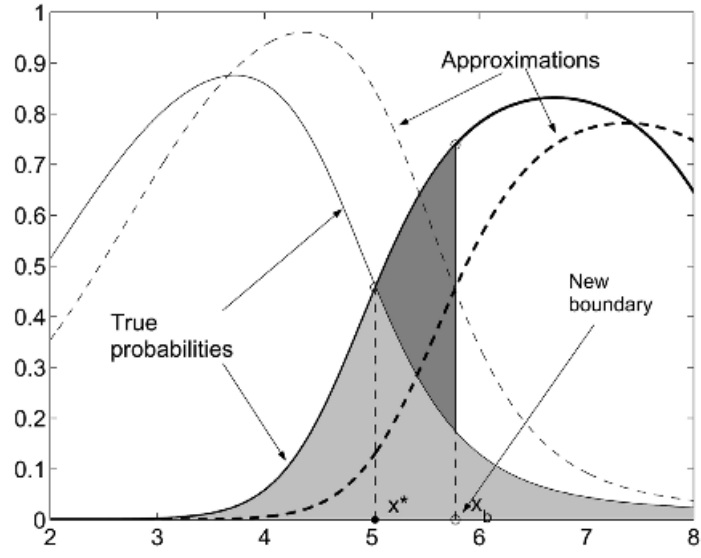


Fig. 9.1 *Plot of two posterior probabilities $P(\omega_i|x)$ (the thin line) and $P(\omega_j|x)$ (the thick line) and their approximations.*

the same value, that is,

$$\hat{P}(\omega_i|x^* + b) = \hat{P}(\omega_j|x^* + b) \tag{9.2}$$

and therefore

$$P(\omega_i|x^* + b) + \varepsilon_i(x_b) = P(\omega_j|x^* + b) + \varepsilon_j(x_b) \tag{9.3}$$

Tumer and Ghosh [279] assume that the posterior probability functions are monotonic in the vicinity of the decision boundary x^*. This is based on the intuition that decision boundaries are typically located in the transition regions between membership and nonmembership to a class, and not in local extrema. Thus we can use a linear approximation, giving

$$P(\omega_i|x^* + b) \approx P(\omega_i|x^*) + bP'(\omega_i|x^*) \tag{9.4}$$

where P' denotes the derivative of P on x. Then Eq. (9.3) becomes

$$P(\omega_i|x^*) + bP'(\omega_i|x^*) + \varepsilon_i(x_b) = P(\omega_j|x^*) + bP'(\omega_j|x^*) + \varepsilon_j(x_b) \tag{9.5}$$

Since for the true boundary x^*, $P(\omega_i|x^*) = P(\omega_j|x^*)$, Eq. (9.5) can be rewritten as

$$b[P'(\omega_j|x^*) - P'(\omega_i|x^*)] = \varepsilon_i(x_b) - \varepsilon_j(x_b) \tag{9.6}$$

Denoting the difference of the derivatives by $s = P'(\omega_j|x^*) - P'(\omega_i|x^*)$, we arrive at

$$b = \frac{\varepsilon_i(x_b) - \varepsilon_j(x_b)}{s} \tag{9.7}$$

The error $\varepsilon_i(x)$ can be broken into bias and zero-mean noise terms

$$\varepsilon_i(x) = \beta_i + \eta_i(x) \tag{9.8}$$

Note, $\varepsilon_i(x)$ is a random variable taking different values for a fixed x depending upon the quality of the approximation.

We shall assume that all estimates are unbiased, that is, $\beta_i = 0$, $i = 1, \ldots, c$. The case of $\beta_i \neq 0$ is followed in Refs. [278–280]. To make the analysis possible, we assume also that the noise terms have the same distribution characteristics across $x \in \Re$, specific for each class, that is, $\eta_i(x) = \eta_i$. Then b is a random variable

$$b = \frac{\eta_i - \eta_j}{s} \tag{9.9}$$

with mean

$$\mathcal{E}(b) = \mathcal{E}\left\{\frac{\eta_i - \eta_j}{s}\right\} = 0 \tag{9.10}$$

and variance

$$\begin{aligned}
\sigma_b^2 &= \mathcal{V}\left\{\frac{\eta_i - \eta_j}{s}\right\} \\
&= \frac{1}{s^2}\mathcal{E}\left\{(\eta_i - \eta_j - \mathcal{E}\{\eta_i - \eta_j\})^2\right\} \\
&= \frac{1}{s^2}\mathcal{E}\left\{(\eta_i - \eta_j)^2\right\} \\
&= \frac{1}{s^2}\left[\mathcal{E}\left\{(\eta_i - 0)^2\right\} + \mathcal{E}\left\{(\eta_j - 0)^2\right\} - 2\mathcal{E}\left\{\eta_i\eta_j\right\}\right] \\
&= \frac{\sigma_i^2 + \sigma_j^2 - 2\text{Cov}\{\eta_i, \eta_j\}}{s^2}
\end{aligned} \tag{9.11}$$

where σ_i^2 and σ_j^2 are the variances of the error for classes ω_i and ω_j, respectively, and $\text{Cov}\{\eta_i, \eta_j\}$ denotes the covariance between the two error terms.

The *added error* incurred by the imprecision of the classification boundary can be derived as a function of the offset b. The dark-shaded region in Figure 9.1, corresponding to the added error, can be approximated by a triangle with a base of length

$$\begin{aligned} P(\omega_j|x^* + b) - P(\omega_i|x^* + b) &= P(\omega_j|x^*) + bP'(\omega_j|x^*) - P(\omega_i|x^*) - bP'(\omega_i|x^*) \\ &= b(P'(\omega_j|x^*) - P'(\omega_i|x^*)) = bs \end{aligned} \tag{9.12}$$

and height b. The area of the triangle is $S_{\triangle}(b) = b^2 s/2$, where s does not depend on b. The added classification error is therefore

$$E_{\text{add}} = \int_{-\infty}^{+\infty} S_{\triangle}(b) p(b)\, db \tag{9.13}$$

where $p(b)$ is the probability density function of b. Substituting the expression for $S_{\triangle}(b)$ in Eq. (9.13), and taking into account that $\mathcal{E}(b) = 0$

$$\begin{aligned} E_{\text{add}} &= \frac{s}{2} \int_{-\infty}^{+\infty} b^2 p(b)\, db = \frac{s}{2} \int_{-\infty}^{+\infty} (b-0)^2 p(b)\, db \\ &= \frac{s}{2} \mathcal{E}\{(b - \mathcal{E}\{b\})^2\} \\ &= \frac{s}{2} \mathcal{V}\{b\} = \frac{s\sigma_b^2}{2} \\ &= \frac{\sigma_i^2 + \sigma_j^2 - 2\text{Cov}\{\eta_i, \eta_j\}}{2s} \end{aligned} \tag{9.14}$$

Note that this result holds for any type of distributions for $\eta_i(x)$, not just Gaussians, as assumed in Refs. [278–280]. Tumer and Ghosh assume also that the interclass noise terms are independent, which brings the expression for the added error (9.14) to

$$E_{\text{add}} = \frac{\sigma_i^2 + \sigma_j^2}{2s} \tag{9.15}$$

For problems with just two classes ($i = 1, j = 2$), the formula is simpler and the assumption of independence is not needed. On the contrary, we assume that the approximations sum to 1,

$$\hat{P}(\omega_1|x) = 1 - \hat{P}(\omega_2|x) \tag{9.16}$$

hence

$$P(\omega_1|x) + \varepsilon_1(x) = 1 - P(\omega_2|x) - \varepsilon_2(x)$$
$$\varepsilon_1(x) = -\varepsilon_2(x) = \varepsilon(x) \tag{9.17}$$

Denote the variance of $\varepsilon(x)$ by σ^2. Then $\text{Cov}\{\eta_1, \eta_2\} = -\sigma^2$. Then Eq. (9.14) becomes

$$E_{\text{add}} = \frac{2\sigma^2}{s} \tag{9.18}$$

Notice also that for this case we have $P'(\omega_1|x^*) = -P'(\omega_2|x^*)$, so

$$s = P'(\omega_2|x^*) - P'(\omega_1|x^*) = -2P'(\omega_1|x^*) \tag{9.19}$$

Then the added error for the classifier for *two classes* is

$$E_{\text{add}} = -\frac{\sigma^2}{P'(\omega_1|x^*)} \tag{9.20}$$

(Note that in our example $P'(\omega_1|x^*) < 0$.)

9.2.2 Added Error of the Ensemble

Consider L classifiers, $D_1 \ldots, D_L$. Each classifier produces estimates of the posterior probabilities. Thus the output of classifier D_m consists of the c estimates $P^m(\omega_1|\mathbf{x}), \ldots, P^m(\omega_c|\mathbf{x})$. The ensemble output is also a set of estimates of the posterior probabilities calculated as the average of the individual estimates

$$P^{\text{ave}}(\omega_k|\mathbf{x}) = \frac{1}{L}\sum_{m=1}^{L} P^m(\omega_k|\mathbf{x}), \quad k = 1, \ldots, c \tag{9.21}$$

A new boundary b_{ave} is found, for which the result in Eq. (9.14) also holds. Therefore

$$E_{\text{add}}^{\text{ave}} = \frac{s\sigma_{\text{ave}}^2}{2} \tag{9.22}$$

where σ_{ave}^2 is the variance of the new boundary b_{ave}. We assume that the classes of interest are again ω_i and ω_j. In other words, we require that the approximations obtained by the averaging of the classifier outputs do not change the relevant classes in the area of the boundary.

To calculate σ^2_{ave}, consider

$$\sigma^2_{\text{ave}} = \mathcal{V}\{b_{\text{ave}}\} = \mathcal{V}\left\{\frac{\bar{\eta}_i - \bar{\eta}_j}{s}\right\} \tag{9.23}$$

The relationship between the two noise terms here is more complex than in Eq. (9.11) because each noise terms $\bar{\eta}_k$, $k = 1, \ldots, c$, are averaged across the L classifiers. Denote by η_k^m the noise term for classifier D_m with respect to class ω_k. We assume again that all η_k^m have zero means, and continue Eq. (9.23) as follows

$$\begin{aligned}\sigma^2_{\text{ave}} &= \frac{1}{s^2}\mathcal{E}\{(\bar{\eta}_i - \bar{\eta}_j)^2\} \\ &= \frac{1}{s^2}\mathcal{E}\{(\bar{\eta}_i)^2 + (\bar{\eta}_j)^2 - 2\bar{\eta}_i\bar{\eta}_j\}\end{aligned} \tag{9.24}$$

Let us take separately the three terms within the expectation brackets of Eq. (9.24).

$$\begin{aligned}\mathcal{E}\{(\bar{\eta}_i)^2\} &= \mathcal{E}\left\{\left(\frac{1}{L}\sum_{m=1}^{L}\eta_i^m\right)^2\right\} \\ &= \frac{1}{L^2}\left[\sum_{m=1}^{L}(\sigma_i^m)^2 + 2\sum_{m=1}^{L-1}\sum_{n=m+1}^{L}\mathcal{E}\{\eta_i^m\eta_i^n\}\right] \\ &= \frac{1}{L^2}\left[\sum_{m=1}^{L}(\sigma_i^m)^2 + \sum_{m=1}^{L}\sum_{n\neq m}\text{Cov}\{\eta_i^m, \eta_i^n\}\right]\end{aligned} \tag{9.25}$$

Similarly,

$$\mathcal{E}\{(\bar{\eta}_j)^2\} = \frac{1}{L^2}\left[\sum_{m=1}^{L}(\sigma_j^m)^2 + \sum_{m=1}^{L}\sum_{n\neq m}\text{Cov}\left\{\eta_j^m, \eta_j^n\right\}\right] \tag{9.26}$$

The third term is

$$\begin{aligned}\mathcal{E}\{\bar{\eta}_i\bar{\eta}_j\} &= \mathcal{E}\left\{\frac{1}{L^2}\sum_{m=1}^{L}\sum_{n=1}^{L}\eta_i^m\eta_j^n\right\} \\ &= \frac{1}{L^2}\left[\sum_{m=1}^{L}\text{Cov}\{\eta_i^m, \eta_j^m\} + \sum_{m=1}^{L}\sum_{n\neq m}\text{Cov}\left\{\eta_i^m, \eta_j^n\right\}\right]\end{aligned} \tag{9.27}$$

Taking the three terms (9.25), (9.26), and (9.27) of Eq. (9.24) together,

$$\sigma_{\text{ave}}^2 = \frac{1}{s^2}\frac{1}{L^2}\left[\sum_{m=1}^{L}(\sigma_i^m)^2 + (\sigma_j^m)^2 - 2\text{Cov}\{\eta_i^m, \eta_j^m\} + \sum_{m=1}^{L}\sum_{n\neq m}\text{Cov}\{\eta_i^m, \eta_i^n\} + \text{Cov}\{\eta_j^m, \eta_j^n\} - 2\text{Cov}\{\eta_i^m, \eta_j^n\}\right] \tag{9.28}$$

According to Eq. (9.14), the individual added errors are

$$E_{\text{add}}^m = \frac{((\sigma_i^m)^2 + (\sigma_j^m)^2 - 2\text{Cov}\{\eta_i^m, \eta_j^m\})}{2s} \tag{9.29}$$

Then the added error of the ensemble in Eq. (9.22) becomes

$$\begin{aligned} E_{\text{add}}^{\text{ave}} &= \frac{s\sigma_{\text{ave}}^2}{2} \\ &= \frac{1}{L^2}\sum_{m=1}^{L}\frac{\left((\sigma_i^m)^2 + (\sigma_j^m)^2 - 2\text{Cov}\{\eta_i^m, \eta_j^m\}\right)}{2s} \\ &\quad + \frac{1}{2L^2 s}\sum_{m=1}^{L}\sum_{n\neq m}\text{Cov}\left\{\eta_i^m, \eta_i^n\right\} + \text{Cov}\left\{\eta_j^m, \eta_j^n\right\} \\ &\quad - 2\text{Cov}\left\{\eta_i^m, \eta_j^n\right\} \\ &= \frac{\bar{E}_{\text{add}}}{L} \\ &\quad + \frac{1}{2L^2 s}\sum_{m=1}^{L}\sum_{n\neq m}\text{Cov}\left\{\eta_i^m, \eta_i^n\right\} + \text{Cov}\left\{\eta_j^m, \eta_j^n\right\} \\ &\quad - 2\text{Cov}\left\{\eta_i^m, \eta_j^n\right\} \end{aligned} \tag{9.30}$$

where $\bar{E}_{\text{add}}$ is the averaged *individual* added error.

9.2.3 Relationship Between the Individual Outputs' Correlation and the Ensemble Error

For independent classifier outputs, all cross-covariances involving D_m and D_n will be zero, and the added error $E_{\text{add}}^{\text{ave}}$ becomes L times smaller than $\bar{E}_{\text{add}}$,

$$E_{\text{add}}^{\text{ave}} = \frac{\bar{E}_{\text{add}}}{L} \tag{9.31}$$

For a check consider the case of identical classifiers. The error terms are $\eta_i^m = \eta_i^n$ for any D_m and D_n from $\mathcal{D}$, and for any $\omega_i \in \Omega$. In this case $\bar{E}_{\text{add}} = E_{\text{add}}$. Then

$$\begin{aligned} E_{\text{add}}^{\text{ave}} &= \frac{E_{\text{add}}}{L} + \frac{1}{2L^2 s} \sum_{m=1}^{L} \sum_{n \neq m} \text{Cov}\{\eta_i^m, \eta_i^n\} + \text{Cov}\{\eta_j^m, \eta_j^n\} \\ &\quad - 2\text{Cov}\{\eta_i^m, \eta_j^n\} \\ &= \frac{E_{\text{add}}}{L} + \frac{(L-1)}{2L^2 s} \sum_{m=1}^{L} \sigma_i^2 + \sigma_j^2 - 2\text{Cov}\{\eta_i, \eta_j\} \\ &= \frac{E_{\text{add}}}{L} + \frac{(L-1)L}{L^2} E_{\text{add}} = E_{\text{add}} \end{aligned} \tag{9.32}$$

As expected, no improvement is gained by combining identical classifiers.

To simplify the analysis of Eq. (9.30), Tumer and Ghosh make the assumptions shown in Figure 9.2.

Under these assumptions, Eq. (9.30) becomes

$$E_{\text{add}}^{\text{ave}} = E_{\text{add}} \left(\frac{1 + \rho(L-1)}{L} \right) \tag{9.35}$$

The correlation coefficient ρ is suggested to be the weighted average of the class-specific correlations. Denote by ρ_i the correlation between η_i^m and η_i^n, averaged across all classifier pairs D_m and D_n. Then

$$\rho = \sum_{i=1}^{c} \hat{P}(\omega_i) \rho_i \tag{9.36}$$

where $\hat{P}(\omega_i)$ is an estimate of the prior probability for class ω_i. For independent classifier outputs, $\rho = 0$, and Eq. (9.31) is recovered from Eq. (9.35). For identical classifiers, $\rho = 1$, and there is no improvement on E_{add}. Equation (9.35) is an elegant result but how realistic is it?

9.2.4 Questioning the Assumptions and Identifying Further Problems

The error terms $\eta_i(x)$ and $\eta_j(x)$, $x \in \Re$, for a particular classifier might not be independent. Typically the estimates of the c posterior probabilities produced by a classifier will sum to 1, that is,

$$\sum_{i=1}^{c} \hat{P}(\omega_i | x) = 1 \tag{9.37}$$

Assumptions

1. The errors in estimating the posterior probabilities for the classes by any single classifier, e.g., D_m, are independent, i.e.,

$$\text{Cov}\{\eta_i^m, \eta_j^m\} = 0, \tag{9.33}$$

   for all classes, $i, j = 1, \ldots, c, i \neq j$.

2. The errors across the classifiers for different classes are independent, i.e., for any two classifiers D_m and D_n, and for all classes $i, j = 1, \ldots, c, i \neq j$,

$$\text{Cov}\{\eta_i^m, \eta_j^n\} = 0. \tag{9.34}$$

3. All variances are the same, i.e., $\sigma_i^m = \sigma, m = 1, \ldots, L, i = 1, \ldots, c$.

4. All correlation coefficients between η_i^m and $\eta_i^n, m, n = 1, \ldots, L$, $m \neq n$, are equal to some ρ_i.

Fig. 9.2 *Assumptions needed for deriving the ensemble added error for the average combination rule.*

Then

$$\sum_{i=1}^{c} P(\omega_i|x) + \eta_i(x) = 1 + \sum_{i=1}^{c} \eta_i(x) = 1 \tag{9.38}$$

hence

$$\sum_{i=1}^{c} \eta_i(x) = 0 \tag{9.39}$$

which contradicts the assumption of interclass independence for any two classes.

Second, the supposed independence of η_i^m and $\eta_j^n, i \neq j, m \neq n$, is also questionable.

Third, it is not clear why the classes' contributions to the correlation, ρ, Eq. (9.36), should be weighted by their prior probabilities.

Another problem that might jeopardize the relationship between $E_{\text{add}}^{\text{ave}}$ and $\bar{E}_{\text{add}}$ is the changing of the relevant classes because of the imprecision of the boundary approximation. The example below illustrates this point.

Consider a three-class problem shown in Figure 9.3. The approximations of the three posterior probabilities are only slightly different from the true probabilities:

$$\hat{P}(\omega_1|x) = P(\omega_1|x) - 0.16$$
$$\hat{P}(\omega_2|x) = P(\omega_2|x) + 0.08$$
$$\hat{P}(\omega_3|x) = P(\omega_3|x) + 0.08 \tag{9.40}$$

However, there is a dramatic change in the decision boundaries as shown in Figure 9.3. The conflict when the true probabilities are concerned is between classes ω_1 and ω_2. Instead of choosing between classes ω_1 and ω_2 at the guessed hypothetical boundary x_b, our classifier will choose class ω_3 whose posterior probability is erroneously estimated to be higher than the other two. Then by using the light gray area in our estimate of the probability of error, we will ignore the large error region incurred by deciding ω_3 instead of ω_1 (shaded in dark gray).

This situation is not too unlikely to occur in practice. Sometimes more than two classes appear to be highly overlapping. For example, in character recognition, around the decision boundary between classes "H" and "K," there might be high posterior probabilities for "B" and "R" as well.

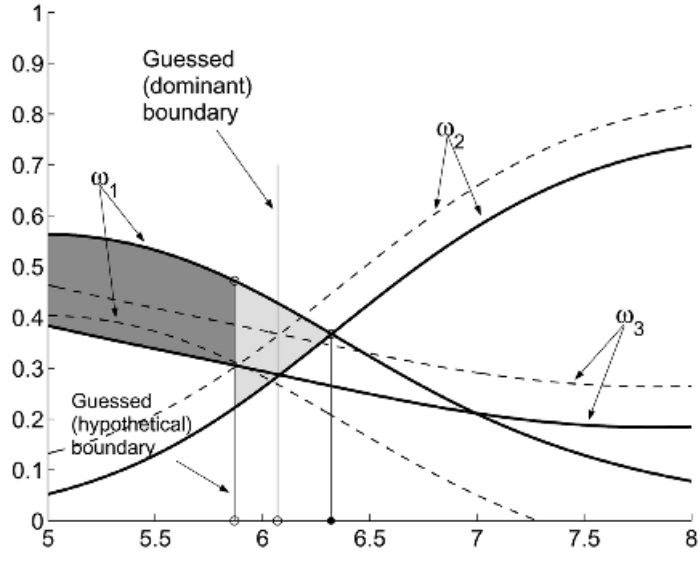


Fig. 9.3 *Changes in the decision boundaries for erroneous approximations of the posterior probabilities*

For the one-dimensional case, $x \in \Re$, the following can happen:

1. Shifting a boundary by a small margin.
2. Creating a boundary where there is none.
3. Failing to detect an existing boundary.

The above results consider only the first case. Tumer and Ghosh note that for more than one dimension, the analysis becomes significantly more complicated [278]. Besides, there is not an easy way to verify Eq. (9.35) or to check whether the assumptions hold. The reason for that is that we can only calculate the correlations between the estimates, that is, $\text{Corr}\{\hat{P}^m(\omega_i|x), P^n(\omega_j|x)\}$, and these are not the correlations between the respective error terms $\eta_i^m(x)$ and $\eta_j^n(x)$. To explain this point, take for example

$$\text{Cov}\{P^m(\omega_i|x), P^n(\omega_i|x)\} = \mathcal{E}\{P(\omega_i|x)^2\} + 2\sigma_i^2 + \text{Cov}\{\eta_i^m(x), \eta_j^n(x)\} \tag{9.41}$$

We are interested in the last term, $\text{Cov}\{\eta_i^m(x), \eta_j^n(x)\}$, but can only measure the total covariance.

9.3 ADDED ERROR FOR THE WEIGHTED MEAN COMBINATION

Fumera and Roli derive the added error for the weighted mean combination rule [165]. The weighted average (weighted mean) finds the ensemble's estimate of $P(\omega_i|x)$ using the individual estimates as

$$P^{\text{wave}}(\omega_i|x) = \sum_{m=1}^{L} w_m P^m(\omega_i|x), \qquad i = 1, \ldots, c \tag{9.42}$$

where w_m are weights, classifier-specific, but constant across x. We require that

$$\sum_{m=1}^{L} w_m = 1, \qquad w_m \geq 0 \tag{9.43}$$

Assumptions 1 and 2 about the individual errors $\eta_i^m(x)$, listed in Figure 9.2, are supposed to hold here as well. However, the important contribution of Fumera and Roli is that there is no assumption of equal variances of the error σ^2, thereby allowing for different added errors of the individual classifiers. The variance of η_i^m is denoted by σ_m^2 for all classes $i = 1, \ldots, c$. Denote also by ρ^{mn} the correlation coefficient between η_i^m and η_i^n, again for all classes, $i = 1, \ldots, c$

$$\rho^{mn} = \frac{\text{Cov}\{\eta_i^m, \eta_i^n\}}{\sigma^2} \tag{9.44}$$

The errors obtained through the weighted average are

$$\bar{\eta}_i = \sum_{m=1}^{L} w_m \eta_i^m \tag{9.45}$$

9.3.1 Error Formula

We start with Eq. (9.24) and take the three terms separately. The first term and the second terms are derived as

$$\begin{aligned}
\mathcal{E}\{(\bar{\eta}_i)^2\} &= \mathcal{E}\left\{\left(\sum_{m=1}^{L} w_m \eta_i^m\right)^2\right\} \\
&= \left[\sum_{m=1}^{L} w_m^2 (\sigma_i^m)^2 + \sum_{m=1}^{L}\sum_{n\neq m} w_m w_n \mathrm{Cov}\{\eta_i^m, \eta_i^n\}\right] \\
&= \left[\sum_{m=1}^{L} w_m^2 \sigma_m^2 + \sum_{m=1}^{L}\sum_{n\neq m} \rho^{mn} w_m w_n \sigma_m \sigma_n\right] \\
&= \mathcal{E}\{(\bar{\eta}_j)^2\}
\end{aligned} \tag{9.46}$$

For the weighted average, the third term of Eq. (9.24) becomes

$$\begin{aligned}
\mathcal{E}\{\bar{\eta}_i \bar{\eta}_j\} &= \mathcal{E}\left\{\sum_{m=1}^{L}\sum_{n=1}^{L} w_m w_n \eta_i^m \eta_j^n\right\} \\
&= \sum_{m=1}^{L} w_m^2 \mathrm{Cov}\{\eta_i^m, \eta_j^m\} + \sum_{m=1}^{L}\sum_{n\neq m} w_m w_n \mathrm{Cov}\{\eta_i^m, \eta_j^n\} \\
&= 0
\end{aligned} \tag{9.47}$$

Then the variance of the averaged estimate is

$$\sigma_{\mathrm{ave}}^2 = \frac{1}{s^2}\left[2\sum_{m=1}^{L} w_m^2 (\sigma^m)^2 + 2\sum_{m=1}^{L}\sum_{n\neq m} \rho^{mn} w_m w_n \sigma_m \sigma_n\right] \tag{9.48}$$

and hence the added error becomes

$$E_{\text{add}}^{\text{wave}} = \frac{s\sigma_{\text{ave}}^2}{2}$$
$$= \frac{1}{s}\left[\sum_{m=1}^{L} w_m^2(\sigma^m)^2 + \sum_{m=1}^{L}\sum_{n\neq m} \rho^{mn} w_m w_n \sigma_m \sigma_n\right] \tag{9.49}$$

The individual added error is the same as for the simple average because we only change the combination rule. For classifier D_m, we have

$$E_{\text{add}}^m = \frac{\sigma_m^2}{s}, \qquad m = 1, \ldots, L \tag{9.50}$$

Thus the added error in the case of weighted average combination becomes

$$E_{\text{add}}^{\text{wave}} = \frac{1}{s}\left[\sum_{m=1}^{L} w_m^2(\sigma^m)^2 + \sum_{m=1}^{L}\sum_{n\neq m} \rho^{mn} w_m w_n \sigma_m \sigma_n\right]$$
$$= \sum_{m=1}^{L} w_m^2 E_{\text{add}}^m + \sum_{m=1}^{L}\sum_{n\neq m} \rho^{mn} w_m w_n \sqrt{E_{\text{add}}^m E_{\text{add}}^n} \tag{9.51}$$

For a check consider again the case of identical classifiers. In this case $\rho^{mn} = 1$, for any m and n, and the individual errors E_{add}^m are all the same, say, E_{add}.

$$E_{\text{add}}^{\text{wave}} = E_{\text{add}}\left[\sum_{m=1}^{L} w_m^2 + 2\sum_{m=1}^{L-1}\sum_{n=m+1}^{L} w_m w_n\right]$$
$$= E_{\text{add}}\left(\sum_{m=1}^{L} w_m\right)^2 = E_{\text{add}} \tag{9.52}$$

As expected, there is no gain in combining identical classifiers i.

Consider the case of independent classifiers, that is, $\rho^{mn} = 0$, for any m and n. The added error of the ensemble in this case is

$$E_{\text{add}}^{\text{wave}} = \sum_{m=1}^{L} w_m^2 E_{\text{add}}^m \tag{9.53}$$

Since $w_m \leq 1$, hence $w_m^2 \leq w_m$, the added error is smaller compared to the averaged *individual* error,

$$\bar{E}_{\text{add}} = \sum_{m=1}^{L} w_m E_{\text{add}}^m \geq E_{\text{add}}^{\text{wave}} \tag{9.54}$$

9.3.2 Optimal Weights for Independent Classifiers

Assume that we know the individual added errors E_{add}^m, $m = 1, \ldots, L$. The question is how do we pick the weights so that the added error of the ensemble is minimal? We use Lagrange multipliers to introduce the constraint of Eq. (9.43) in the minimization. Take the derivative,

$$\frac{\partial}{\partial w_k}\left[E_{\text{add}}^{\text{ave}} + \lambda\left(\sum_{m=1}^{L} w_m - 1\right)\right] = 2w_k E_{\text{add}}^k + \lambda \tag{9.55}$$

Then solve simultaneously the following $L+1$ equations

$$2w_k E_{\text{add}}^k + \lambda = 0, \qquad k = 1, \ldots, L \tag{9.56}$$

$$\sum_{m=1}^{L} w_m = 1 \tag{9.57}$$

to find the optimal combination weights to be

$$w_k = (E_{\text{add}}^k)^{-1}\left[\sum_{m=1}^{L}(E_{\text{add}}^m)^{-1}\right]^{-1}, \qquad k = 1, \ldots, L \tag{9.58}$$

Thus the worse the classifier, the lower the weight. If all weights were equal, that is, if we used the simple average, we recover Eq. (9.31). Clearly, if the individual errors are different, the weighted average with the optimal coefficients will produce a lower added error than the simple average.

Substituting the optimal weights into Eq. (9.53), the added error for independent classifiers combined through the weighted average becomes

$$E_{\text{add}}^{\text{wave}} = \sum_{k=1}^{L}\left\{\frac{1}{E_{\text{add}}^k}\left[\sum_{m=1}^{L}\frac{1}{E_{\text{add}}^m}\right]^{-1}\right\}^2 E_{\text{add}}^k \tag{9.59}$$

$$= \frac{\sum_{k=1}^{L}(1/E_{\text{add}}^k)}{(\sum_{m=1}^{L}(1/E_{\text{add}}^m))^2} = \left(\sum_{m=1}^{L}\frac{1}{E_{\text{add}}^m}\right)^{-1} \tag{9.60}$$

Fumera and Roli [165] proceed to analyze the difference in the added error if we used equal weights, $w_k = 1/L$ (the simple average combination) and the added error derived as Eq. (9.60), that is,

$$\Delta E_{\text{add}} = \frac{1}{L^2}\sum_{m=1}^{L} E_{\text{add}}^m - \left(\sum_{m=1}^{L}\frac{1}{E_{\text{add}}^m}\right)^{-1} \tag{9.61}$$

Example: Gain of Weighted Average Over Simple Average. Take 10 *independent* classifiers with the same range of the added errors (0.16 to 0.24), and consider the following four cases:

- *Case 1*: *Random*.
  Added errors: 0.16 0.19 0.23 0.16 0.17 0.18 0.18 0.21 0.18 0.24
- *Case 2*: *50/50*.
  Added errors: 0.16 0.16 0.16 0.16 0.16 0.24 0.24 0.24 0.24 0.24
- *Case 3*: *One best standing out*.
  Added errors: 0.16 0.24 0.24 0.24 0.24 0.24 0.24 0.24 0.24 0.24
- *Case 4*: *All good but one*.
  Added errors: 0.16 0.16 0.16 0.16 0.16 0.16 0.16 0.16 0.16 0.24

The added errors for the simple average, weighted average, and the difference ΔE_{add}, are given below.

Case	$E_{\text{add}}^{\text{ave}}$	$E_{\text{add}}^{\text{wave}}$	ΔE_{add}
1	0.0190	0.0187	0.000339
2	0.0200	0.0192	0.000800
3	0.0232	0.0229	0.000343
4	0.0168	0.0166	0.000248

The largest difference was found for case 2, 50/50.

An interesting continuation of this study is to define an index describing the pattern of the errors of the ensemble. Fumera and Roli [165] suggest

$$\delta_E = \frac{\bar{E}_{\text{add}}^{\text{tr}} - \min_m E_{\text{add}}^m}{\max_m E_{\text{add}}^m - \min_m E_{\text{add}}^m} \tag{9.62}$$

where $\bar{E}_{\text{add}}^{\text{tr}}$ is the trimmed mean of the added errors, dropping the largest and the smallest errors. They call δ_E the *degree of performance imbalancing*.

Fumera and Roli [281] also analyze ΔE_{add} and the improvement on the single best classifier for ensembles of $L \geq 3$ classifiers. Their results can be summarized as shown in Figure 9.4. The conclusion of this study is that weighted average might not be as good as expected! The authors found that for large ensembles the advantage of weighted averaging over simple averaging disappears. Besides, in weighted averaging we have to estimate the L weights, which is a potential source of error, and, as the authors suggest, may cancel the anyway small advantage. There is a glimpse of hope for the weighted average though. In all the experiments reported in Ref. [281], weighted average was better, albeit marginally, than the simple average.

9.4 ENSEMBLE ERROR FOR NORMAL AND UNIFORM DISTRIBUTIONS OF THE OUTPUTS

The framework for this section is defined by the assumptions displayed in Figure 9.5 [282,283].

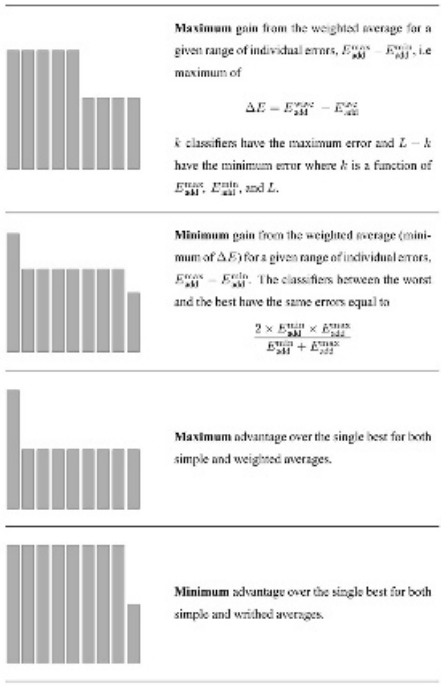


Fig. 9.4 *Examples of the distributions of the individual errors of the ensemble members defining the most and least favorable cases for the weighted average combination method. $L = 9$ classifiers.*

We discuss two distributions of $\eta(\mathbf{x})$: a normal distribution with mean 0 and variance σ^2 (we take σ to vary between 0.1 and 1), and a uniform distribution spanning the interval $[-b, +b]$ (b varies from 0.1 to 1). Thus $d_{j,1}(\mathbf{x})$ is a random variable with normal or uniform distribution with mean p and variance σ^2, and $d_{j,2}(\mathbf{x})$ is a random variable with normal or uniform distribution with mean $1 - p$ and the same variance, σ^2.

Simple fusion methods are the most obvious choice when constructing a multiple classifier system [11,149,159,241,280], that is, the support for class ω_i, $\mu_i(\mathbf{x})$, yielded by the team is

$$\mu_i(\mathbf{x}) = \mathcal{F}(d_{1,i}(\mathbf{x}), \ldots, d_{L,i}(\mathbf{x})), \qquad i = 1, 2 \tag{9.64}$$

where $\mathcal{F}$ is the chosen fusion method. Here we study the minimum, maximum, average, median, and majority vote.

Recall that for the majority vote we first "harden" the individual decisions by assigning class label ω_1 if $d_{j,1}(\mathbf{x}) > 0.5$, and ω_2 if $d_{j,1}(\mathbf{x}) \leq 0.5$, $j = 1, \ldots, L$. Then the class label most represented among the L (label) outputs is chosen as

Assumptions

1. There are two possible classes $\Omega = \{\omega_1, \omega_2\}$.
2. All classifiers produce soft class labels, $d_{j,i}(\mathbf{x}) \in [0, 1]$, $i = 1, 2$, $j = 1, \ldots, L$, where $d_{j,i}(\mathbf{x})$ is an estimate of the posterior probability $P(\omega_i|\mathbf{x})$ by classifier D_j for an input $\mathbf{x} \in \Re^n$. We consider the case where for any $\mathbf{x}$, $d_{j,1}(\mathbf{x}) + d_{j,2}(\mathbf{x}) = 1$, $j = 1, \ldots, L$.
3. A single point $\mathbf{x} \in \Re^n$ is considered at a time. Without loss of generality, we assume that the true posterior probability is $P(\omega_1|\mathbf{x}) = p > 0.5$. Thus, the Bayes-optimal class label for $\mathbf{x}$ is ω_1 and a classification error occurs if label ω_2 is assigned.
4. The classifiers commit independent and identically distributed errors in estimation $P(\omega_1|\mathbf{x})$, i.e.,

$$d_{j,1}(\mathbf{x}) = P(\omega_1|\mathbf{x}) + \eta(\mathbf{x}) = p + \eta(\mathbf{x}), \tag{9.63}$$

and respectively $d_{j,2}(\mathbf{x}) = 1 - p - \eta(\mathbf{x})$.

Fig. 9.5 *Assumptions needed for deriving the ensemble error for normal and uniform distributions of the individual estimates.*

the final label for $\mathbf{x}$. The *oracle* model is analyzed as well. In this model, if at least one of the classifiers produces the correct class label, then the team produces the correct class label too.

Denote by P_j the output of classifier D_j for class ω_1, that is, $P_j = d_{j,1}(\mathbf{x})$ and let

$$\hat{P}_1 = \mathcal{F}(P_1, \ldots, P_L) \tag{9.65}$$

be the fused estimate of $P(\omega_1|\mathbf{x})$. By assumption, the posterior probability estimates for ω_2 are $1 - P_j, j = 1, \ldots, L$. The same fusion method $\mathcal{F}$ is used to find the fused estimate of $P(\omega_2|\mathbf{x})$,

$$\hat{P}_2 = \mathcal{F}(1 - P_1, \ldots, 1 - P_L) \tag{9.66}$$

According to the assumptions, we regard the individual estimates P_j as independent identically distributed random variables, such that $P_j = p + \eta_j$, with probability density functions (pdf) $f(y)$, $y \in \Re$ and cumulative distribution functions (cdf) $F(t)$, $t \in \Re$. Then $\hat{P}_1$ is a random variable too with a pdf $f_{\hat{P}_1}(y)$ and cdf $F_{\hat{P}_1}(t)$.

For a single classifier, the average and the median fusion models will result in $\hat{P}_1 + \hat{P}_2 = 1$. The higher of the two estimates determines the class label. The oracle and the majority vote make decisions on the class label outputs, and we can stipulate

that $\hat{P}_1 = 1, \hat{P}_2 = 0$ if class ω_1 is assigned, and $\hat{P}_1 = 0, \hat{P}_2 = 1$ for class ω_2. Thus, it is necessary and sufficient to have $\hat{P}_1 > 0.5$ to label $\mathbf{x}$ in ω_1 (the correct label). The probability of error, given $\mathbf{x}$, denoted P_e, is

$$P_e = P(\text{error}|\mathbf{x}) = P(\hat{P}_1 \leq 0.5) = F_{\hat{P}_1}(0.5) = \int_0^{0.5} f_{\hat{P}_1}(y)\, dy \tag{9.67}$$

for the single best classifier, average, median, majority vote and the oracle.

For the minimum and the maximum rules, however, the sum of the fused estimates is not necessarily one. The class label is then decided by the maximum of $\hat{P}_1$ and $\hat{P}_2$. Thus, an error will occur if $\hat{P}_1 \leq \hat{P}_2$,[35]

$$P_e = P(\text{error}|\mathbf{x}) = P(\hat{P}_1 \leq \hat{P}_2) \tag{9.68}$$

for the minimum and the maximum.

The two distributions considered are

- Normal distribution, $\hat{P}_1 \sim N(p, \sigma^2)$. We denote by $\Phi(z)$ the cumulative distribution function of $N(0, 1)$.[36] Thus, the cumulative distribution function for the normal distribution considered here is

$$F(t) = \Phi\left(\frac{t-p}{\sigma}\right) \tag{9.69}$$

- Uniform distribution within $[p - b, p + b]$, that is,

$$f(y) = \begin{cases} \dfrac{1}{2b}, & y \in [p-b, p+b]; \\ 0, & \text{elsewhere}, \end{cases}$$

$$F(t) = \begin{cases} 0, & t \in (-\infty, p-b); \\ \dfrac{t-p+b}{2b}, & t \in [p-b, p+b]; \\ 1, & t > p+b. \end{cases} \tag{9.70}$$

Clearly, using these two distributions, the estimates of the probabilities might fall outside the interval [0, 1]. We can accept this, and justify our viewpoint by the following argument. Suppose that p is not a probability but the *amount of support* for ω_1. The support for ω_2 will be again $1 - p$. In estimating p, we do not have to restrict P_j values within the interval [0, 1]. For example, a neural network (or *any* classifier for that matter) trained by minimizing the squared error between its output and the

[35] We note that since P_1 and P_2 are continuous-valued random variables, the inequalities can be written with or without the equal sign, that is, $\hat{P}_1 > 0.5$ is equivalent to $\hat{P}_1 \geq 0.5$, and so on.

[36] Available in tabulated form or from any statistical package.

zero–one (class label) target function produces an estimate of the posterior probability for that class [27]. Thus, depending on the parameters and the transition functions, a neural network output (that approximates p) might be greater than 1 and also might be negative. We take the L values (in $\Re$) and fuse them by Eqs. (9.65) and (9.66) to get $\hat{P}_1$. The same rule applies, that is, ω_1 is assigned by the ensemble if $\hat{P}_1 > \hat{P}_2$. Then we calculate the *probability* of error P_e as $P(\hat{P}_1 \leq \hat{P}_2)$. This calculation does not require in any way that P_j values are probabilities or are within the unit interval.

9.4.1 Individual Error

Since $F_{\hat{P}_1}(t) = F(t)$, the error of a single classifier for the normal distribution is

$$P_e = \Phi\left(\frac{0.5 - p}{\sigma}\right) \tag{9.71}$$

and for the uniform distribution

$$P_e = \frac{0.5 - p + b}{2b} \tag{9.72}$$

9.4.2 Minimum and Maximum

These two fusion methods are considered together because, as shown in Section 9.1.1, they are identical for $c = 2$ classes and any number of classifiers L.

Substituting $\mathcal{F} = \max$ in Eq. (9.65), the team's support for ω_1 is $\hat{P}_1 = \max_j \{P_j\}$. The support for ω_2 is therefore $\hat{P}_2 = \max_j \{1 - P_j\}$. A classification error will occur if

$$\max_j \{P_j\} < \max_j \{1 - P_j\} \tag{9.73}$$

$$p + \max_j \{\eta_j\} < 1 - p - \min_j \{\eta_j\} \tag{9.74}$$

$$\eta_{\max} + \eta_{\min} < 1 - 2p \tag{9.75}$$

The probability of error for minimum and maximum is

$$P_e = P(\eta_{\max} + \eta_{\min} < 1 - 2p) \tag{9.76}$$

$$= F_{\eta_s}(1 - 2p) \tag{9.77}$$

where $F_{\eta_s}(t)$ is the cdf of the random variable $s = \eta_{\max} + \eta_{\min}$. For the normally distributed P_j values, η_j are also normally distributed with mean 0 and variance σ^2. However, we cannot assume that $\eta_{\max}$ and $\eta_{\min}$ are independent and analyze their

sum as another normally distributed variable because these are *order statistics* and $\eta_{\min} \leq \eta_{\max}$. We have not attempted a solution for the normal distribution case.

For the uniform distribution, we follow an example taken from Ref. [284] where the pdf of the midrange $(\eta_{\min} + \eta_{\max})/2$ is calculated for L observations. We derive $F_{\eta_s}(t)$ to be

$$F_{\eta_s}(t) = \begin{cases} \frac{1}{2}\left(\frac{t}{2b} + 1\right)^L, & t \in [-2b, 0]; \\ 1 - \frac{1}{2}\left(1 - \frac{t}{2b}\right)^L, & t \in [0, 2b] \end{cases} \tag{9.78}$$

Noting that $t = 1 - 2p$ is always negative,

$$P_e = F_{\eta_s}(1 - 2p) = \frac{1}{2}\left(\frac{1 - 2p}{2b} + 1\right)^L \tag{9.79}$$

9.4.3 Mean

The average fusion method gives $\hat{P}_1 = 1/L \sum_{j=1}^{L} P_j$. If $P_1, \ldots, P_L$ are normally distributed (and independent!), then $\hat{P}_1 \sim N\left(p, \sigma^2/L\right)$. The probability of error for this case is

$$P_e = P(\hat{P}_1 < 0.5) = \Phi\left(\frac{\sqrt{L}(0.5 - p)}{\sigma}\right) \tag{9.80}$$

The calculation of P_e for the case of uniform distribution is not that straightforward. We can assume that the sum of L independent variable will result in a variable of approximately normal distribution. The higher the L, the more accurate the approximation. Knowing that the variance of the uniform distribution for P_j is $b^2/3$, we can assume $\hat{P} \sim N\left(p, b^2/3L\right)$. Then

$$P_e = P(\hat{P}_1 < 0.5) = \Phi\left(\frac{\sqrt{3L}(0.5 - p)}{b}\right) \tag{9.81}$$

9.4.4 Median and Majority Vote

These two fusion methods are pooled because they are identical for the current setup (see Section 9.1.2).

Since only two classes are considered, we restrict our choice of L to odd numbers only. An even L is inconvenient for at least two reasons. First, the majority vote might tie. Second, the theoretical analysis of a median that is calculated as the average of the $(L/2)$ and $(L/2 + 1)$ order statistics is cumbersome.

For the median fusion method

$$\hat{P}_1 = \text{med}\{P_1, \ldots, P_L\} = p + \text{med}\{\eta_1, \ldots, \eta_L\} = p + \eta_m \tag{9.82}$$

Then the probability of error is

$$P_e = P(p + \eta_m < 0.5) = P(\eta_m < 0.5 - p) = F_{\eta_m}(0.5 - p) \tag{9.83}$$

where F_{η_m} is the cdf of η_m. From the order statistics theory [284],

$$F_{\eta_m}(t) = \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} F_\eta(t)^j [1 - F_\eta(t)]^{L-j} \tag{9.84}$$

where $F_\eta(t)$ is the distribution of η_j, that is, $N(0, \sigma^2)$ or uniform in $[-b, b]$. We can now substitute the two cdf, to obtain the respective P_e

- for the normal distribution

$$P_e = \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} \Phi\left(\frac{0.5 - p}{\sigma}\right)^j \left[1 - \Phi\left(\frac{0.5 - p}{\sigma}\right)\right]^{L-j} \tag{9.85}$$

- for the uniform distribution

$$P_e = \begin{cases} 0, & p - b > 0.5; \\ \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} \left(\frac{0.5 - p + b}{2b}\right)^j \left[1 - \frac{0.5 - p + b}{2b}\right]^{L-j}, & \text{otherwise.} \end{cases} \tag{9.86}$$

The majority vote will assign the wrong class label, ω_2, to $\mathbf{x}$ if at least $L + 1/2$ classifiers vote for ω_2. The probability that a single classifier is wrong is given by Eq. (9.71) for the normal distribution and Eq. (9.72) for the uniform distribution. Denote this probability by P_s. Since the classifiers are independent, the probability that at least $L + 1/2$ are wrong is calculated by the binomial formula

$$P_e = \sum_{j=\frac{L+1}{2}}^{L} \binom{L}{j} P_s^j [1 - P_s]^{L-j} \tag{9.87}$$

By substituting for P_s from Eq. (9.71) and Eq. (9.72), we recover Eqs. (9.85) and (9.86) for the normal and the uniform distribution, respectively.

9.4.5 Oracle

The probability of error for the oracle is

$$P_e = P(\text{all incorrect}) = F(0.5)^L \tag{9.88}$$

For the normal distribution

$$P_e = \Phi\left(\frac{0.5-p}{\sigma}\right)^L \tag{9.89}$$

and for the uniform distribution

$$P_e = \begin{cases} 0, & p-b > 0.5; \\ \left(\dfrac{0.5-p+b}{2b}\right)^L, & \text{otherwise.} \end{cases} \tag{9.90}$$

Table 9.1 displays in a compact form the results for the two distributions, the single classifier and the six fusion methods.

9.4.6 Example

A direct comparison between the errors in Table 9.1 is hardly possible, except for the single classifier and the oracle, where the preference is known anyway. Figure 9.6 plots the classification error of the single classifier and the team, calculated by the respective equations for normally distributed P_js. The top two plots depict P_e against σ for a fixed $p = 0.6$, and the bottom two plots depict P_e against p for a fixed

TABLE 9.1 The Theoretical Error P_e for the Single Classifier and the Six Fusion Methods.

Method	P_e for Normal Distribution	P_e for Uniform Distribution ($p-b<0.5$)
Single classifier	$\Phi\left(\frac{0.5-p}{\sigma}\right)$	$\frac{0.5-p+b}{2b}$
Minimum/maximum	–	$\frac{1}{2}\left(\frac{1-2p}{2b}+1\right)^L$
Average	$\Phi\left(\frac{\sqrt{L}(0.5-p)}{\sigma}\right)$	$\Phi\left(\frac{\sqrt{3L}(0.5-p)}{b}\right)$
Median/majority vote	$\sum_{j=\frac{L+1}{2}}^{L}\binom{L}{j}\times\Phi\left(\frac{0.5-p}{\sigma}\right)^j \times\left[1-\Phi\left(\frac{0.5-p}{\sigma}\right)\right]^{L-j}$	$\sum_{j=\frac{L+1}{2}}^{L}\binom{L}{j}\times\left(\frac{0.5-p+b}{2b}\right)^j \times\left[1-\frac{0.5-p+b}{2b}\right]^{L-j}$
Oracle	$\Phi\left(\frac{0.5-p}{\sigma}\right)^L$	$\left(\frac{0.5-p+b}{2b}\right)^L$

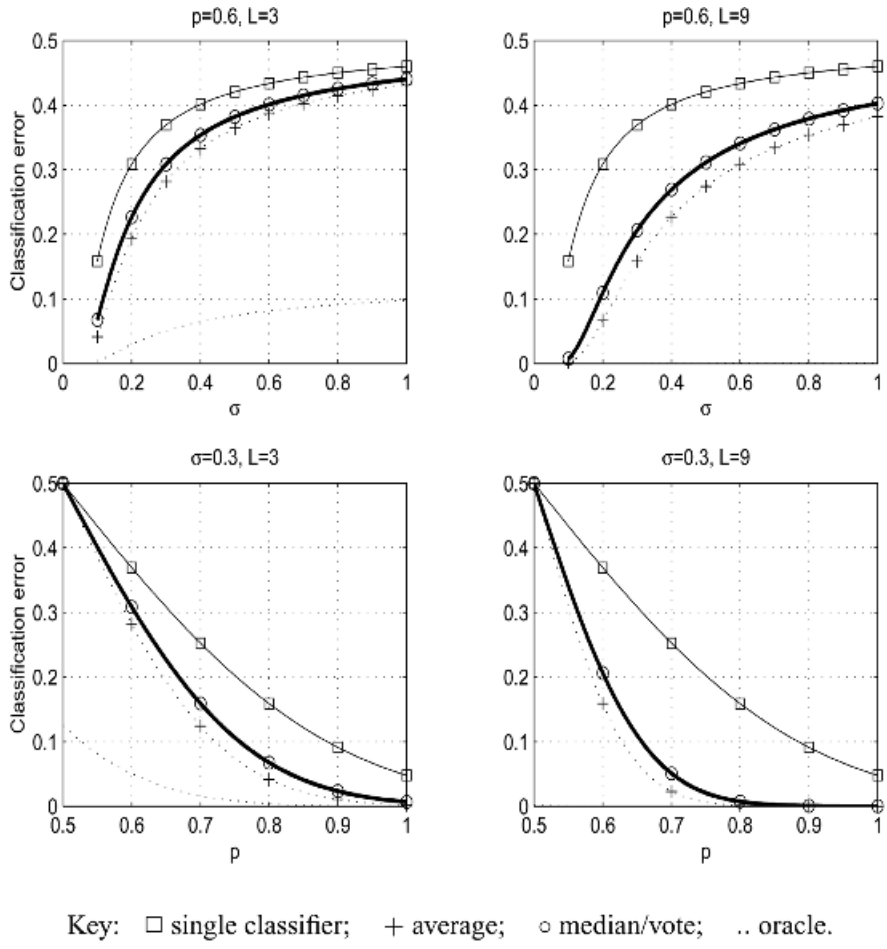


Fig. 9.6 *P_e for normally distributed P_js.*

$\sigma = 0.3$. Figure 9.7 displays the results for uniformly distributed P_js. The top two plots depict P_e against b for a fixed $p = 0.6$, and the bottom two plots depict P_e against p for a fixed $b = 0.8$.

The results can be summarized as:

1. *Expected results*. These are well documented in the literature on classifier combination.
   (a) The individual error is higher than the error of any of the fusion methods.
   (b) The oracle model (an abstraction) is the best of all. For $L = 9$, the oracle error rate is approximately zero.
   (c) The more classifiers we have in the team, the lower the error. Recall that the classifiers are assumed to be independent, which can hardly be achieved in real-life problems.

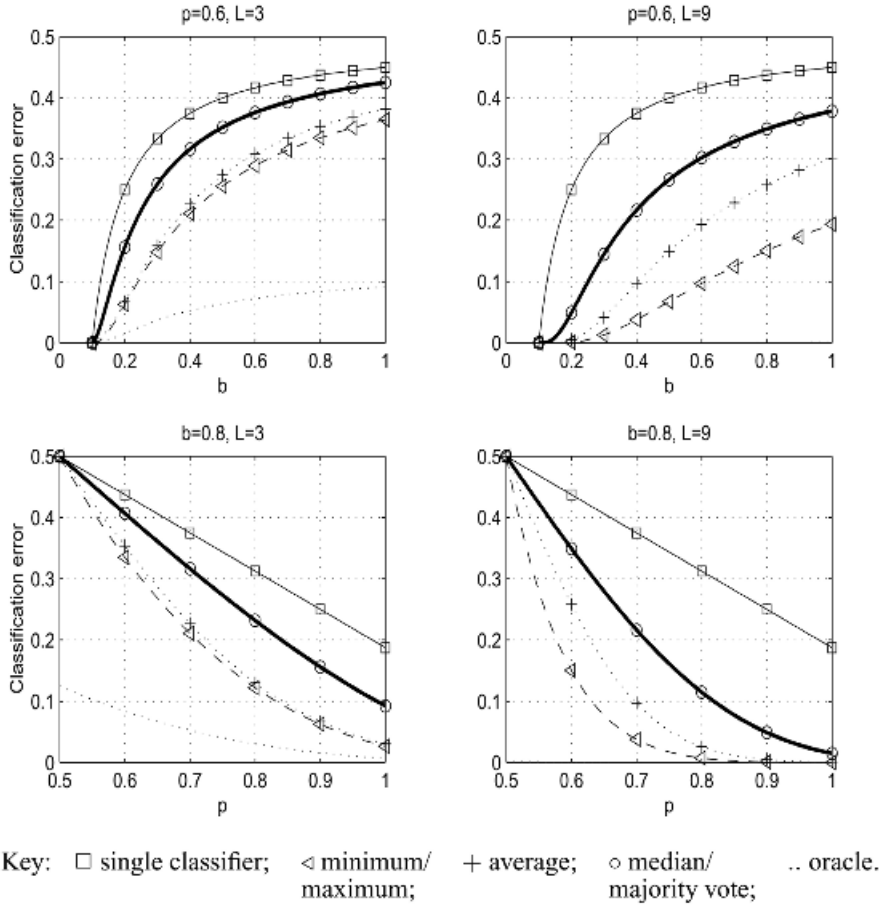


Fig. 9.7 *P_e for uniformly distributed P_js.*

2. More interesting findings from this example.
   (a) The average and the median/vote methods have approximately the same performance for normally distributed P_j, but are different for the uniform distribution, the average being the better of the two.
   (b) Contrary to some experimental evidence published elsewhere, the average method is outperformed by the minimum/maximum method. This observation is based on the uniform distribution model only. Unfortunately, the theoretical calculation of P_e for the minimum/maximum method in the case of normally distributed P_js is not easy, and we cannot draw a parallel with the average in this case.

It is claimed in the literature that combination methods are less important than the diversity of the team. However, given a set of classifiers, the only way to extract the

most of it is to pick a good combination method. Indeed, for normally distributed errors, the fusion methods gave very similar performance, but for the uniformly distributed error, the methods differed significantly, especially for higher L. For example, the top right plot in Figure 9.7 shows that P_e for a single classifier of 40.0 percent can be reduced to 26.7 percent by the median or majority vote, 14.9 percent by the average, and 6.7 percent by minimum or maximum fusion. This example comes in support of the idea that combination methods are also relevant in combining classifiers.

Similar analyses can be carried out for distributions other than Normal or uniform. Kittler and Alkoot [285] and Chen and Cheng [286] studied the behavior of the sum, median, and vote combination methods for nonnormal distributions. Their conclusions suggest that while for symmetrical distributions the methods perform similarly, nonsymmetrical distributions might lead to large differences.

10

Diversity in Classifier Ensembles

Common intuition suggests that the classifiers in the ensemble should be as accurate as possible and should not make coincident errors. As simple as this sounds, there are a variety of diversity measures and their relationship with the ensemble accuracy is ambiguous. The methods for building ensembles, which rely on inducing diversity in an intuitive manner, are very successful. Even weakening the individual classifiers for the sake of better diversity appears to be an excellent ensemble building strategy, unequivocally demonstrated by AdaBoost. Ironically, trying to measure diversity and use it explicitly in the process of building the ensemble does not share the success of the implicit methodologies. This chapter introduces measures of diversity in an attempt to present the philosophy, the state of the art, and the possible directions of the studies in classifier diversity.

10.1 WHAT IS DIVERSITY?

If we have a perfect classifier that makes no errors, then we do not need an ensemble. If, however, the classifier does make errors, then we seek to complement it with another classifier, which makes errors on different objects. The diversity of the classifier outputs is therefore a vital requirement for the success of the ensemble. Intuitively, we want the ensemble members to be as correct as possible, and in case they make errors, these errors should be on different objects. In practice, it appeared to be difficult to define a single measure of diversity, and even more difficult to relate that

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

measure to the ensemble performance in a neat and expressive dependency. The most useful ideas often drift across sciences and branches thereof.

10.1.1 Diversity in Biology

Suppose that we are interested in the height of adult gorillas in a certain region of Africa. Consider a population π with a probability measure P associated with it. The measure P defines the distribution of heights for the population. A comprehensive study on diversity in life sciences by Rao [287] gives the following axiomatic definition of a diversity measure.

Let $(\mathcal{X}, \mathcal{B})$ be a measurable space, and let $\mathcal{P}$ be a convex set of probability measures defined on it.[37] A function $H(\cdot)$ mapping $\mathcal{P}$ onto the real line is said to be a *measure of diversity* if it satisfies the following conditions

C1: $H(p) \geq 0$, for any $p \in \mathcal{P}$ and $H(p) = 0$ iff p is degenerate.

C2: H is a concave function of p.[38]

The concavity condition ensures that any mixture of two populations has a higher diversity than the average of the two individual diversities. $H(p_i)$ is the diversity within a population π_i characterized by the probability measure P_i. To quantify diversity we need a measure of difference or distance between pairs of objects, X_1 and X_2, from the population. The distance, $\zeta(X_1, X_2)$, could be any function that satisfies the axioms for distance (nonnegativity, symmetry, and a version of the triangle inequality). We can use the Euclidean distance for quantitative variables and a matching type of function for qualitative variables, that is,

$$\zeta(X_1, X_2) = \begin{cases} 1, & \text{if } X_1 \neq X_2 \\ 0, & \text{if } X_1 = X_2. \end{cases} \tag{10.1}$$

Rao defines $H(p_i)$ to be the averaged *difference* between two randomly picked individuals in the population π_i according to the probability measure p_i

$$H(P_i) = \int_{X_1, X_2} \zeta(X_1, X_2) p_i(X_1) p_i(X_2)\, dX_1 dX_2 \tag{10.2}$$

If the two individuals are drawn from two different populations π_i and π_j, then the total diversity will be

$$H(p_i, p_j) = \int_{X_1, X_2} \zeta(X_1, X_2) p_i(X_1) p_j(X_2)\, dX_1 dX_2 \tag{10.3}$$

[37] Convexity means that for any $p_1, p_2 \in \mathcal{P}$, and for any $t \in [0, 1]$, $tp_1 + (1-t)p_2 \in \mathcal{P}$.

[38] The concavity here means that for any $p_1, p_2 \in \mathcal{P}$, and for any $t \in [0, 1]$,

$$H(tp_1 + (1-t)p_2) \geq tH(p_1) + (1-t)H(p_2)$$

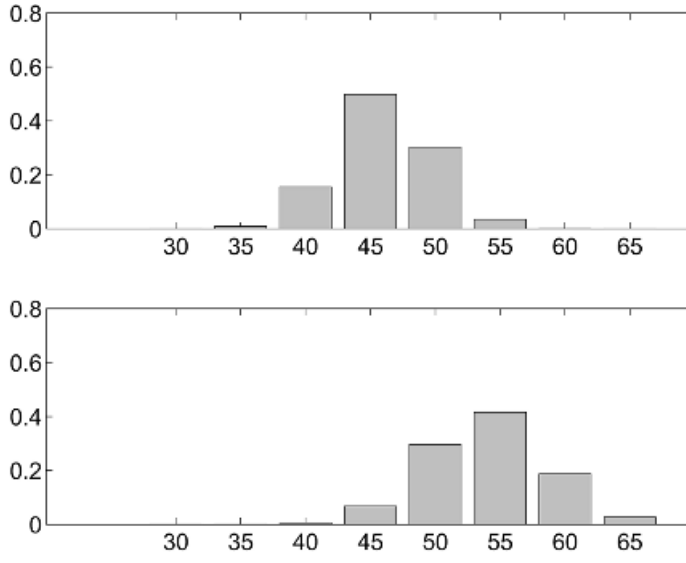


Fig. 10.1 *Distribution of the numbers of eggs in a single brood by female green iguanas in two different populations. The top plot gives the distribution for π_1 and the bottom plot gives the distribution for π_2.*

The *dissimilarity* between populations π_i and π_j is

$$D_{ij} = H(p_i, p_j) - \frac{1}{2}(H(p_i) + H(p_j)) \tag{10.4}$$

The concavity of H guarantees that D_{ij} will be positive for any two populations and their probability measures. This dissimilarity is based on taking out the diversity coming from each population and leaving only the "pure" diversity due to mixing the two populations.

Example: Diversity Between Two Populations. Let π_1 and π_2 be two populations of green iguanas and X be the number of eggs a female iguana produces in a single brood.[39] Owing to climate differences in their habitat, the distributions of X differ for the two populations. The two distributions are illustrated in Figure 10.1 and shown numerically in Table 10.1.

If we take $\zeta(X_1, X_2) = |X_1 - X_2|$ to be the distance between two individuals X_1 and X_2, then using Eqs. (10.2) to (10.4), we get $H(p_1) = 4.0504$, $H(p_2) = 5.0058$, $H(p_1, p_2) = 8.4046$, and $D_{1,2} = 3.8764$.

Consider a data set $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ and an ensemble, $\mathcal{D} = \{D_1, \ldots, D_L\}$. Each classifier suggests a class label from $\Omega = \{\omega_1, \ldots, \omega_c\}$ for every data point $\mathbf{z}_j$. A *population* is defined to be a collection of *objects*. The objects within the classifier

[39] The data follows an example in Ref. [288].

TABLE 10.1 Probability Distributions for Populations π_1 and π_2.

X	25	30	35	40	45	50	55	60
$p_1(X)$, in [%]	0.01	0.91	15.51	49.81	30.21	3.46	0.07	0.00
$p_2(X)$, in [%]	0.00	0.01	0.50	6.76	29.60	41.63	18.79	2.72

ensemble framework can be either the classifiers or the data points. If a population consists of classifiers, then we have N populations of classifiers, one per data point. Since we are interested in the diversity within each population, $(H(p_i))$, we can calculate N diversity values and average across $\mathbf{Z}$.[40] Alternatively, we may regard the output of each classifier as a population with N individuals in it, and look for diversity between the L populations. A pairwise measure of this quantity is D_{ij} in Section 10.4. The $L(L-1)/2$ pairwise measures can be averaged to get an overall diversity value for the ensemble.

10.1.2 Diversity in Software Engineering

A major issue in software engineering is the reliability of software. Multiple programs (called versions) can be run in parallel in the hope that if one or more fail, the others will compensate for it by producing correct outputs. It is tempting to assume that the errors of the versions will be independent if the versions are created independently. It appears, however, that independently created versions fail together on "difficult" assignments and run correctly together on "easy" assignments [289–291].

While the biological model does not have a clear-cut analogy with classifier ensembles, the multiversion software model does. The programs (versions) correspond to the classifiers in the ensemble and the inputs correspond to the points in the feature space.

The model proposed in Ref. [290] and developed further in Refs. [291,292] considers a set of programs and a set of inputs. The quantity of interest, which underpins several measures of diversity (discussed later), is the probability that two randomly selected programs will fail on a randomly chosen input.

10.1.3 Statistical Measures of Relationship

For convenience, below is the list of the three major types of classifier outputs (see Chapter 4):

- *Oracle*. For a given data set $\mathbf{Z}$, classifier D_i produces an output vector $\mathbf{y}_i$ such that

$$y_{ij} = \begin{cases} 1, & \text{if } D_i \text{ classifies object } \mathbf{z}_j \text{ correctly,} \\ 0, & \text{otherwise.} \end{cases} \tag{10.5}$$

[40] This is the idea of the KW diversity measure discussed later.

Clearly oracle outputs are only possible for a labeled data set and their use is limited to the design stage of the classifiers and the ensemble.

- *Label*. The output of the classifier is a label from Ω.
- *Soft outputs*. Classifier D_i gives c values of support, $d_{i,j}, j = 1, \ldots, c$, for the classes.

Various measures of the relationship between two variables can be found in the statistical literature [293].

10.1.3.1 Correlation. Correlation coefficients can be calculated for pairs of classifiers using soft outputs. There will be c coefficients for each pair of classifiers, one for each class. To get a single measure of diversity, the correlations can be averaged across classes.

Correlation can be calculated for a pair of classifiers with oracle outputs because we can treat the two values (0 and 1) numerically. To illustrate the calculation, consider a table of the joined (oracle) outputs of classifiers D_i and D_j as shown in Table 4.4 (for convenience we reproduce the table here as Table 10.2). The entries in the table are the probabilities for the respective pair of correct/incorrect outputs.

The correlation between two binary classifier outputs is

$$\rho_{i,j} = \frac{ad - bc}{\sqrt{(a+b)(c+d)(a+c)(b+d)}} \tag{10.6}$$

10.1.3.2 The Q Statistic. Using Table 10.2, Yule's Q statistic [294] for classifiers D_i and D_j is

$$Q_{i,j} = \frac{ad - bc}{ad + bc} \tag{10.7}$$

For statistically *independent* classifiers, $Q_{i,j} = 0$. Q varies between -1 and 1. Classifiers that tend to recognize the *same* objects correctly will have positive values of Q. For any two classifiers, Q and ρ have the same sign, and it can be proved that $|\rho| \leq |Q|$.

10.1.3.3 Interrater Agreement, κ. A statistic developed as a measure of interrater reliability, called κ, can be used when different raters (here classifiers) assess subjects (here $\mathbf{z}_j$) to measure the level of agreement while correcting for chance [20].

TABLE 10.2 The 2 × 2 Relationship Table with Probabilities.

	D_j correct (1)	D_j wrong (0)
D_i correct (1)	a	b
D_i wrong (0)	c	d

Total, $a + b + c + d = 1$.

For c class labels, κ is defined on the $c \times c$ coincidence matrix M of the two classifiers. The entry $m_{k,s}$ of M is the proportion of the data set, which D_i labels as ω_k and D_j labels as ω_s. The agreement between D_i and D_j is given by

$$\kappa_{i,j} = \frac{\sum_k m_{kk} - \text{ABC}}{1 - \text{ABC}} \tag{10.8}$$

where $\sum_k m_{kk}$ is the observed agreement between the classifiers and "ABC" is "agreement-by-chance"

$$\text{ABC} = \sum_k \left(\sum_s m_{k,s} \right) \left(\sum_s m_{s,k} \right) \tag{10.9}$$

Low values of κ signify higher disagreement and hence higher diversity. If calculated on the 2×2 joined oracle output space using probabilities,

$$\kappa_{i,j} = \frac{2(ac - bd)}{(a+b)(c+d) + (a+c)(b+d)} \tag{10.10}$$

10.2 MEASURING DIVERSITY IN CLASSIFIER ENSEMBLES

Along with borrowing ideas for diversity measures across the disciplines, there are measures of diversity developed specifically for classifier ensembles.

10.2.1 Pairwise Measures

These measures, and the ones discussed hitherto, consider a pair of classifiers at a time. An ensemble of L classifiers will produce $L(L-1)/2$ pairwise diversity values. To get a single value we average across all pairs.

10.2.1.1 The Disagreement Measure. The disagreement measure is probably the most intuitive measure of diversity between a pair of classifiers. For the oracle outputs, this measure is equal to the probability that the two classifiers will disagree on their decisions, that is,

$$D_{i,j} = b + c \tag{10.11}$$

Without calling it a disagreement measure, this statistic has been used in the literature for analysing classifier ensembles [243,295].

The disagreement measure is equivalent to the total diversity $H(p_i, p_j)$ within the biological interpretation of diversity. Suppose that π_i and π_j are two populations produced by classifiers D_i and D_j. Consider oracle outputs and a new space with

four elements: 00, 01, 10, and 11. Using as a distance measure $\zeta(m, n)$, given by Eq. (10.1),

$$H(p_i, p_j) = \zeta(1, 1) \times a + \zeta(1, 0) \times b + \zeta(0, 1) \times c + \zeta(0, 0) \times d = b + c$$
$$= D_{i,j} \tag{10.12}$$

$H(p_i, p_j)$ is the expectation of the disagreement between classifiers D_i and D_j in the space of their joint oracle outputs.

10.2.1.2 The Double-Fault Measure. The double fault measure is another intuitive choice, as it gives the probability of classifiers D_i and D_j both being wrong,

$$DF_{i,j} = d \tag{10.13}$$

Ruta and Gabrys [296] note that DF is a *nonsymmetrical* diversity measure, that is, if we swap the 0s and the 1s, DF will no longer have the same value. This measure is based on the concept that it is more important to know when *simultaneous errors* are committed than when both classifiers are correct. Thus the measure is related by design to the ensemble performance.

All diversity measures introduced so far are *pairwise*. To get an overall value for the ensemble we can average across all pairs. There are also nonpairwise measures as discussed below.

10.2.2 Nonpairwise Measures

The measures of diversity introduced below consider all the classifiers together and calculate directly one diversity value for the ensemble.

10.2.2.1 The Entropy Measure E. Intuitively, the ensemble is most diverse for a particular $\mathbf{z}_j \in \mathbf{Z}$ when $\lfloor L/2 \rfloor$ of the votes are 0s (1s) and the other $L - \lfloor L/2 \rfloor$ votes are 1s (0s).[41] If they all were 0s or all were 1s, there is no disagreement, and the classifiers cannot be deemed diverse. One possible measure of diversity based on this concept is

$$E = \frac{1}{N} \frac{2}{L-1} \sum_{j=1}^{N} \min \left\{ \left(\sum_{i=1}^{L} y_{j,i} \right), \left(L - \sum_{i=1}^{L} y_{j,i} \right) \right\} \tag{10.14}$$

E varies between 0 and 1, where 0 indicates no difference and 1 indicates the highest possible diversity. Let all classifiers have the same individual accuracy p. Then while value 0 is achievable for any number of classifiers L and any p, the value 1 can only be attained for $p \in \left[\frac{L-1}{2L}, \frac{L+1}{2L} \right]$.

[41] $\lfloor a \rfloor$ is the "floor" function. It returns the largest integer smaller than a. $\lceil a \rceil$ is the "ceiling" function. It returns the smallest integer greater than a.

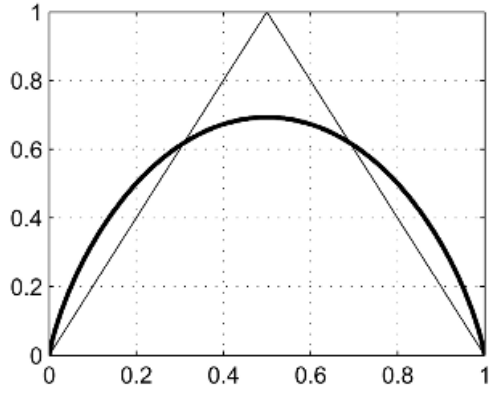


Fig. 10.2 *The two entropies E(a) (thin line) and $E_{CC}(a)$ (thick line) plotted versus a.*

It should be noted that E is not a standard entropy measure because it does not use the logarithm function. A classical version of this measure is proposed by Cunningham and Carney [297] (we denote it here as E_{CC}). Taking the expectation over the whole feature space, letting the number of classifiers tend to infinity ($L \to \infty$), and denoting by a the proportion of 1s (correct outputs) in the ensemble, the two entropies are

$$E(a) = \frac{1}{2}\min\{a, 1-a\} \tag{10.15}$$

$$E_{CC}(a) = -a\log(a) - (1-a)\log(1-a) \tag{10.16}$$

Figure 10.2 plots the two entropies versus a. The two measures are equivalent up to a (nonlinear) monotonic transformation. This means that they will have a similar pattern of relationship with the ensemble accuracy.

10.2.2.2 Kohavi–Wolpert Variance. Kohavi and Wolpert derived a decomposition formula for the error rate of a classifier [231]. Consider a classifier model. Let y be the predicted class label for $\mathbf{x}$. The variance of y across different training sets that were used to build the classifier is defined to be

$$\text{variance}_x = \frac{1}{2}\left(1 - \sum_{i=1}^{c} P(y = \omega_i|\mathbf{x})^2\right) \tag{10.17}$$

The variance (10.17) is the *Gini index* for the distribution of the classifier output regarded as a set of probabilities, $P(y = \omega_1|\mathbf{x}), \ldots, P(y = \omega_c|\mathbf{x})$. The variance also coincides with the biological definition of diversity *within the population* $H(p)$ in Eq. (10.2) for $\zeta(m, n)$ as in Eq. (10.1).

We use the general idea for calculating the variance for each $\mathbf{z}_j$ in the following way. We look at the variability of the predicted class label for $\mathbf{z}_j$ for the given training set using the classifier models $D_1, \ldots, D_L$. Instead of Ω, here we consider just two possible classifier outputs: correct and incorrect. In the Kohavi–Wolpert framework, $P(y = \omega_i|\mathbf{z}_j)$ is estimated as an average across different data sets. In our case, $P(y = 1|\mathbf{z}_j)$ and $P(y = 0|\mathbf{z}_j)$ will be obtained as an average across the set of classifiers $\mathcal{D}$, that is,

$$\hat{P}(y = 1|\mathbf{z}_j) = \frac{Y(\mathbf{z}_j)}{L} \quad \text{and} \quad \hat{P}(y = 0|\mathbf{z}_j) = \frac{L - Y(\mathbf{z}_j)}{L} \tag{10.18}$$

where $Y(\mathbf{z}_j)$ is the number of correct votes for $\mathbf{z}_j$ among the L classifiers, that is,

$$Y(\mathbf{z}_j) = \sum_{i=1}^{L} y_{i,j}$$

Substituting Eq. (10.18) into Eq. (10.17),

$$\text{variance}_x = \frac{1}{2}(1 - \hat{P}(y = 1|\mathbf{z}_j)^2 - \hat{P}(y = 0|\mathbf{z}_j)^2) \tag{10.19}$$

and averaging over the whole of $\mathbf{Z}$, we set the *KW* measure of diversity to be

$$KW = \frac{1}{NL^2} \sum_{j=1}^{N} Y(\mathbf{z}_j)(L - Y(\mathbf{z}_j)) \tag{10.20}$$

Interestingly, *KW* differs from the averaged disagreement measure D_{av} by a coefficient, that is,

$$KW = \frac{L - 1}{2L} D_{\text{av}} \tag{10.21}$$

(The proof of the equivalence is given in Appendix 10A.)

10.2.2.3 Measurement of Interrater Agreement, κ, For $L > 2$. If we denote $\bar{p}$ to be the average individual classification accuracy, then [20]

$$\kappa = 1 - \frac{\frac{1}{L}\sum_{j=1}^{N} Y(\mathbf{z}_j)(L - Y(\mathbf{z}_j))}{N(L - 1)\bar{p}(1 - \bar{p})} \tag{10.22}$$

It is easy to see that κ is related to *KW* and D_{av} as follows

$$\kappa = 1 - \frac{L}{(L - 1)\bar{p}(1 - \bar{p})} KW = 1 - \frac{1}{2\bar{p}(1 - \bar{p})} D_{\text{av}} \tag{10.23}$$

Note that κ is not equal to the averaged pairwise kappa, $\kappa_{i,j}$ in Eq. (10.10).

10.2.2.4 The Measure of "difficulty" θ. The idea for this measure came from a study by Hansen and Salamon [298]. We define a discrete random variable X taking values in $\{0/L, 1/L, \ldots, 1\}$ and denoting the proportion of classifiers in $\mathcal{D}$

that correctly classify an input $\mathbf{x}$ drawn randomly from the distribution of the problem. To estimate the probability mass function of X, the L classifiers in $\mathcal{D}$ are run on the data set $\mathbf{Z}$.

Figure 10.3 shows three possible histograms of X for $L = 7$ and $N = 100$ data points. We assumed that all classifiers have individual accuracy $p = 0.6$. The leftmost plot shows the histogram if the seven classifiers were independent. In this case the discrete random variable $X \times L$ has a Binomial distribution ($p = 0.6$, $n = L$). The middle plot shows seven identical classifiers. They all recognize correctly the *same* 60 points and misclassify the remaining 40 points in $\mathbf{Z}$. The rightmost plot corresponds to the case of negatively dependent classifiers. They recognize different subsets of $\mathbf{Z}$. The figures in the histogram are calculated so that the sum of all correct votes is $L \times p \times N = 7 \times 0.6 \times 100 = 420$. That is, if m_i denotes the number of data points for $X = i/L$, in all three histograms, $\sum_{i=1}^{L} im_i = 420$.

Hansen and Salamon [298] talk about a pattern of "difficulty" of the points in the feature space, which in our case is represented by the histogram over $\mathbf{Z}$. If the same points have been *difficult* for all classifiers, and the other points have been *easy* for all classifiers, we obtain a plot similar to the middle one (no diversity in the ensemble). If the points that were difficult for some classifiers were easy for other classifiers, the distribution of X is as the one on the right. Finally, if each point is equally difficult for all classifiers, the distribution on the left is the most likely one. Diverse ensembles $\mathcal{D}$ will have smaller variance of X (right plot). Ensembles of similar classifiers will have high variance, as in the pattern in the middle plot. Let the three variables X in Figure 10.3*a*, 10.3*b*, and 10.3*c* be X_a, X_b, and X_c, respectively. The three variances are

$$\theta_a = \mathrm{Var}(X_a) = 0.034 \qquad \theta_b = \mathrm{Var}(X_b) = 0.240, \qquad \theta_c = \mathrm{Var}(X_c) = 0.004$$

Therefore we define the measure of *difficulty* θ to be $\mathrm{Var}(X)$. For convenience we can scale θ linearly into [0, 1], taking $p(1-p)$ as the highest possible value. The higher the value of θ, the worse the classifier ensemble. Ideally, $\theta = 0$, but this is

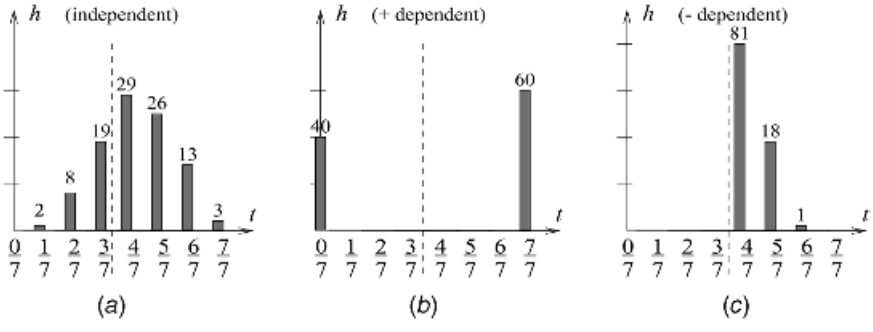


Fig. 10.3 *Patterns of "difficulty" for three classifier ensembles with $L = 7$, $p = 0.6$, and $N = 100$. The dashed line is the majority vote border. The histograms show the number of points (out of 100) that are correctly labeled by i of the L classifiers. The x-axis is "proportion correct," that is, i/L.*

an unrealistic scenario. More often, real classifiers are positively dependent and will exhibit patterns similar to Figure 10.3*b*.

10.2.2.5 Generalized Diversity. Partridge and Krzanowski [291] consider a random variable Y, expressing the proportion of classifiers (out of L) that are incorrect (or *fail*) on a randomly drawn object $\mathbf{x} \in \Re^n$. Denote by p_i the probability that $Y = i/L$, that is, the probability that *exactly* i out of the L classifiers fail on a randomly chosen input. (Note that $Y = 1 - X$, where X was introduced for θ.) Denote by $p(i)$ the probability that i *randomly chosen* classifiers will fail on a randomly chosen $\mathbf{x}$. Suppose that two classifiers are randomly picked from $\mathcal{D}$. Partridge and Krzanowski argue that maximum diversity occurs when failure of one of these classifiers is accompanied by correct labeling by the other classifier. In this case the probability of both classifiers failing is $p(2) = 0$. Minimum diversity occurs when a failure of one classifier is always accompanied by a failure of the other classifier. Then the probability of both classifiers failing is the same as the probability of one randomly picked classifier failing, that is, $p(1)$. Using

$$p(1) = \sum_{i=1}^{L} \frac{i}{L} p_i, \qquad \text{and} \qquad p(2) = \sum_{i=1}^{L} \frac{i}{L} \frac{(i-1)}{(L-1)} p_i \tag{10.24}$$

the generalization diversity measure GD is defined as

$$GD = 1 - \frac{p(2)}{p(1)} \tag{10.25}$$

GD varies between 0 (minimum diversity when $p(2) = p(1)$) and 1 (maximum diversity when $p(2) = 0$).

10.2.2.6 Coincident Failure Diversity. Coincident failure diversity is a modification of GD also proposed by Partridge and Krzanowski [291].

$$CFD = \begin{cases} 0, & p_0 = 1.0; \\ \dfrac{1}{1 - p_0} \sum_{i=1}^{L} \dfrac{L - i}{L - 1} p_i, & p_0 < 1. \end{cases} \tag{10.26}$$

This measure is designed so that it has a minimum value of 0 when all classifiers are always correct or when all classifiers are simultaneously either correct or wrong. Its maximum value 1 is achieved when all misclassifications are unique, that is, when at most one classifier will fail on any randomly chosen object.

Note that GD and CFD originated from the software reliability literature. Partridge and Krzanowski propose also the *distinct failure diversity*, DFD [292]. For oracle outputs we do not know exactly which (wrong) class label has been assigned if the classifier makes an error. If all classifiers choose the same wrong label, then this is a sign of low diversity. Conversely, classifiers suggesting

different wrong labels is a sign of diversity. For oracle outputs the two cases are indistinguishable as the wrong labels will all be coded by one value.

Various other diversity measures have been proposed [299–301], which once again illustrates the *diversity of diversity*.

10.3 RELATIONSHIP BETWEEN DIVERSITY AND ACCURACY

The general anticipation is that diversity measures will be helpful in designing the individual classifiers, the ensemble, and choosing the combination method. For this to be possible, there should be a relationship between diversity and the ensemble performance.

10.3.1 Example

To investigate the hypothetical relationship between diversity and the ensemble accuracy we recall the example in Section 4.2.2. We generated all possible distributions of correct/incorrect votes for 10 objects and three classifiers, such that each classifier recognizes exactly 6 of the 10 objects (individual accuracy $p = 0.6$). The 28 possible vote distributions are displayed in Table 4.3. The accuracy of the ensemble of three classifiers, each of accuracy 0.6, varied between 0.4 and 0.9. The two limit distributions were called the "pattern of success" and the "pattern of failure," respectively.

Example: Calculation of Diversity Measures. We take as our example row 27 from Table 4.3. The 10 objects are so distributed that $P_{\text{maj}} = 0.5$ even though all three classifiers have accuracy $p = 0.6$. For an easier reference, the distribution of the votes (correct/wrong) of row 27 of Table 4.3 is duplicated in Table 10.3.

The three tables with the probability estimates for the classifier pairs are shown in Table 10.4.

TABLE 10.3 A Distribution of the Votes of Three Classifiers (Row 27 from Table 4.3).

D_1, D_2, D_3	111	101	011	001	110	100	010	000
Frequency	3	0	0	3	2	1	1	0

TABLE 10.4 The Three Pairwise Tables for the Distribution in Table 10.3.

D_1, D_2		D_1, D_3		D_2, D_3	
0.5	0.1	0.3	0.3	0.3	0.3
0.1	0.3	0.3	0.1	0.3	0.1

The pairwise measures of diversity are calculated as follows

$$Q_{1,2} = \frac{5 \times 3 - 1 \times 1}{5 \times 3 + 1 \times 1} = \frac{7}{8}$$

$$Q_{1,3} = Q_{2,3} = \frac{3 \times 1 - 3 \times 3}{3 \times 1 + 3 \times 3} = -\frac{1}{2}$$

$$Q = \frac{1}{3}\left(\frac{7}{8} - \frac{1}{2} - \frac{1}{2}\right) = -\frac{1}{24} \approx \mathbf{-0.04} \tag{10.27}$$

$$\rho_{1,2} = \frac{5 \times 3 - 1 \times 1}{\sqrt{(5+1)(1+3)(5+1)(1+3)}} = \frac{7}{12}$$

$$\rho_{1,3} = \rho_{2,3} = \frac{3 \times 1 - 3 \times 3}{(5+1)(1+3)} = -\frac{1}{4}$$

$$\rho = \frac{1}{3}\left(\frac{7}{12} - \frac{1}{4} - \frac{1}{4}\right) = \frac{1}{36} \approx \mathbf{0.03} \tag{10.28}$$

$$D = \frac{1}{3}((0.1 + 0.1) + (0.3 + 0.3) + (0.3 + 0.3))$$

$$= \frac{7}{15} \approx \mathbf{0.47} \tag{10.29}$$

$$DF = \frac{1}{3}(0.3 + 0.3 + 0.1) = \frac{1}{6} \approx \mathbf{0.17} \tag{10.30}$$

The nonpairwise measures KW, κ, and E are calculated by

$$KW = \frac{1}{10 \times 3^2}(3 \times (1 \times 2) + 2 \times (2 \times 1) + 1 \times (1 \times 2) + 1 \times (1 \times 2))$$

$$= \frac{7}{45} \approx \mathbf{0.16} \tag{10.31}$$

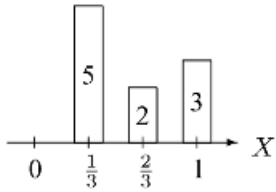
$$\kappa = 1 - \frac{D}{2 \times 0.6 \times (1 - 0.6)} = 1 - \frac{7/15}{12/25}$$

$$= \frac{1}{36} \approx \mathbf{0.03} \tag{10.32}$$

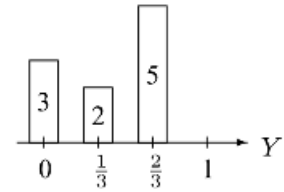
$$E = \frac{1}{10} \times \frac{2}{(3-1)} \times (3 \times \min\{3, 0\} + 3 \times \min\{1, 2\}$$

$$+ 2 \times \min\{2, 1\} + 1 \times \min\{1, 2\} + 1 \times \min\{1, 2\})$$

$$= \frac{7}{10} = \mathbf{0.70} \tag{10.33}$$



X	0	1/3	2/3	1
$Pr(X)$	0	0.5	0.2	0.3



Y	0	1/3	2/3	1
$Pr(Y)$	0.3	0.2	0.5	0

Fig. 10.4 *The frequencies and the probability mass functions of the variables X and Y needed for calculating diversity measures θ, GD, and CFD.*

The distribution of the random variables X and Y needed for θ, GD, and CFD are depicted in Figure 10.4.

The mean of X is 0.6, and the mean of $Y(p(1))$ is 0.4. The respective measures are calculated as follows:

$$\theta = \text{Var}(X) = (1/3 - 0.6)^2 \times 0.5 + (2/3 - 0.6)^2 \times 0.2 + (1 - 0.6)^2 \times 0.3$$
$$= \frac{19}{225} \approx \mathbf{0.08} \tag{10.34}$$

$$p(2) = \frac{2}{3} \times \frac{(2-1)}{(3-1)} \times 0.5 = \frac{1}{6}$$
$$GD = 1 - \frac{1/6}{0.4} = \frac{7}{12} \approx \mathbf{0.58} \tag{10.35}$$

$$CFD = \frac{1}{1 - 0.3}\left(\frac{(3-1)}{(3-1)} \times 0.2 + \frac{(3-2)}{(3-1)} \times 0.5\right)$$
$$= \frac{9}{14} \approx \mathbf{0.64} \tag{10.36}$$

Calculated in this way, the values of the 10 diversity measures for all distributions of classifier votes from Table 4.3 are shown in Table 10.5. To enable cross-referencing, the last column of the table shows the majority vote accuracy of the ensemble, P_{maj}. The rows are arranged in the same order as in Table 4.3.

With 10 objects, it is not possible to model pairwise independence. The table of probabilities for this case will contain $a = 0.36$, $b = c = 0.24$, and $d = 0.16$. To use 10 objects, we have to round so that $a = 0.4$, $b = c = 0.2$, and $d = 0.2$, but instead

TABLE 10.5 The 10 Diversity Measures and the Majority Vote Accuracy, P_{maj}, for the 28 Distributions of Classifier Votes in Table 4.3. The Ensembles Separated with Lines Are: (Row 1) Pattern of Success, (Row 13) Identical Classifiers, and (Row 28) Pattern of Failure.

No.	Q	ρ	D	DF	KW	κ	E	θ	GD	CFD	P_{maj}
1	−0.50	−0.25	0.60	0.10	0.20	−0.25	0.90	0.04	0.75	0.90	0.9
2	0.33	0.17	0.40	0.20	0.13	0.17	0.60	0.11	0.50	0.75	0.8
3	−0.22	−0.11	0.53	0.13	0.18	−0.11	0.80	0.06	0.67	0.83	0.8
4	−0.67	−0.39	0.67	0.07	0.22	−0.39	1.0	0.02	0.83	0.90	0.8
5	−0.56	−0.39	0.67	0.07	0.22	−0.39	1.0	0.02	0.83	0.90	0.8
6	0.88	0.58	0.20	0.30	0.07	0.58	0.30	0.17	0.25	0.50	0.7
7	0.51	0.31	0.33	0.23	0.11	0.31	0.50	0.13	0.42	0.64	0.7
8	0.06	0.03	0.47	0.17	0.16	0.03	0.70	0.08	0.58	0.75	0.7
9	−0.04	0.03	0.47	0.17	0.16	0.03	0.70	0.08	0.58	0.75	0.7
10	−0.50	−0.25	0.60	0.10	0.20	−0.25	0.90	0.04	0.75	0.83	0.7
11	−0.39	−0.25	0.60	0.10	0.20	−0.25	0.90	0.04	0.75	0.83	0.7
12	−0.38	−0.25	0.60	0.10	0.20	−0.25	0.90	0.04	0.75	0.83	0.7
13	1.0	1.0	0.00	0.40	0.00	1.0	0.00	0.24	0.00	0.00	0.6
14	0.92	0.72	0.13	0.33	0.04	0.72	0.20	0.20	0.17	0.30	0.6
15	0.69	0.44	0.27	0.27	0.09	0.44	0.40	0.15	0.33	0.50	0.6
16	0.56	0.44	0.27	0.27	0.09	0.44	0.40	0.15	0.33	0.50	0.6
17	0.33	0.17	0.40	0.20	0.13	0.17	0.60	0.11	0.50	0.64	0.6
18	0.24	0.17	0.40	0.20	0.13	0.17	0.60	0.11	0.50	0.64	0.6
19	0.00	0.17	0.40	0.20	0.13	0.17	0.60	0.11	0.50	0.64	0.6
20	−0.22	−0.11	0.53	0.13	0.18	−0.11	0.80	0.06	0.67	0.75	0.6
21	−0.11	−0.11	0.53	0.13	0.18	−0.11	0.80	0.06	0.67	0.75	0.6
22	−0.21	−0.11	0.53	0.13	0.18	−0.11	0.80	0.06	0.67	0.75	0.6
23	−0.33	−0.11	0.53	0.13	0.18	−0.11	0.80	0.06	0.67	0.75	0.6
24	0.88	0.58	0.20	0.30	0.07	0.58	0.30	0.17	0.25	0.30	0.5
25	0.51	0.31	0.33	0.23	0.11	0.31	0.50	0.13	0.42	0.50	0.5
26	0.06	0.03	0.47	0.17	0.16	0.03	0.70	0.08	0.58	0.64	0.5
27	−0.04	0.03	0.47	0.17	0.16	0.03	0.70	0.08	0.58	0.64	0.5
28	0.33	0.17	0.40	0.20	0.13	0.17	0.60	0.11	0.50	0.50	0.4

of 0, this gives a value of Q

$$Q = \frac{0.08 - 0.04}{0.08 + 0.04} = \frac{1}{3}$$

In this sense, closest to independence are rows 2, 17, and 23.

10.3.2 Relationship Patterns

It is not easy to spot by eye in Table 10.5 any relationship between diversity and accuracy for any of the diversity measures. To visualize a possible relationship we give a scatterplot of diversity Q_{av} versus improvement in Figure 10.5.

Each point in the figure corresponds to a classifier ensemble. The x-coordinate is the diversity value, Q_{av}, averaged for the three pairs of classifiers, (D_1, D_2), (D_1, D_3), and (D_2, D_3). The y-value is the improvement $P_{maj} - p$. Since

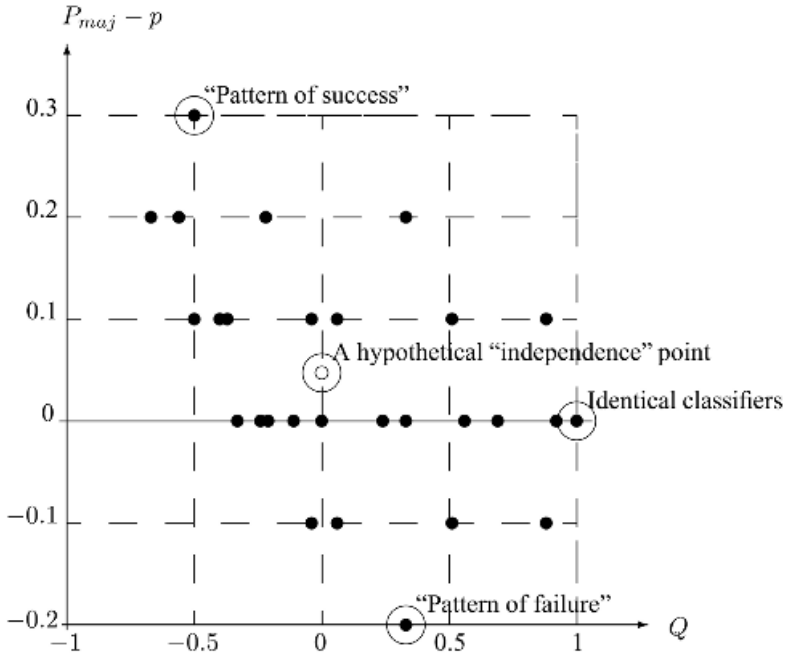


Fig. 10.5 *Improvement on the individual accuracy (P_{maj}-p) versus Q_{av}.*

each classifier has individual accuracy 0.6, the y-value is simply $P_{\text{maj}} - 0.6$. The scatter of the points does not support the intuition about the relationship between diversity and accuracy. If there was a relationship, the points would be distributed along a straight or a curved line with a backward slop, indicating that the lower the values of Q (high diversity), the greater the improvement. What the figure shows is that the best ensemble is not found for the minimum Q_{av} and the worst ensemble is not found for the maximum Q_{av}. The patterns of success and failure occur for values of Q_{av} within the span of possible values for this experiment.

The hypothetical independence point shows only mild improvement of about 0.05 on the individual accuracy p, much smaller than the improvement of 0.30 corresponding to the pattern of success. Note, however, that values of $Q_{\text{av}} = 0$ are associated with improvement of 0.10 and also with a decline of the performance by 0.10. For the hypothetical independence point all three pairwise Q are 0, that is, $Q_{1,2} = Q_{1,3} = Q_{2,3} = 0$, while for the other points at the same Q_{av}, the individual diversities just add to 0. This suggests that a single measure of diversity might not be accurate enough to capture all the relevant diversity in the ensemble.

The relationship patterns were not substantially different for the other measures either. To populate the scatterplots with points we repeated the simulation but took $N = 30$ objects ($L = 3$ classifiers, each of accuracy $p = 0.6$), which gave a total of

563 ensembles. Figure 10.6 shows the scatterplots of the improvement versus diversity for the ten measures. As the figure shows, the measures are not unequivocally related to the improvement. The plots will be the same if we plotted P_{maj} instead of the improvement because we will only shift the y axis by a constant.

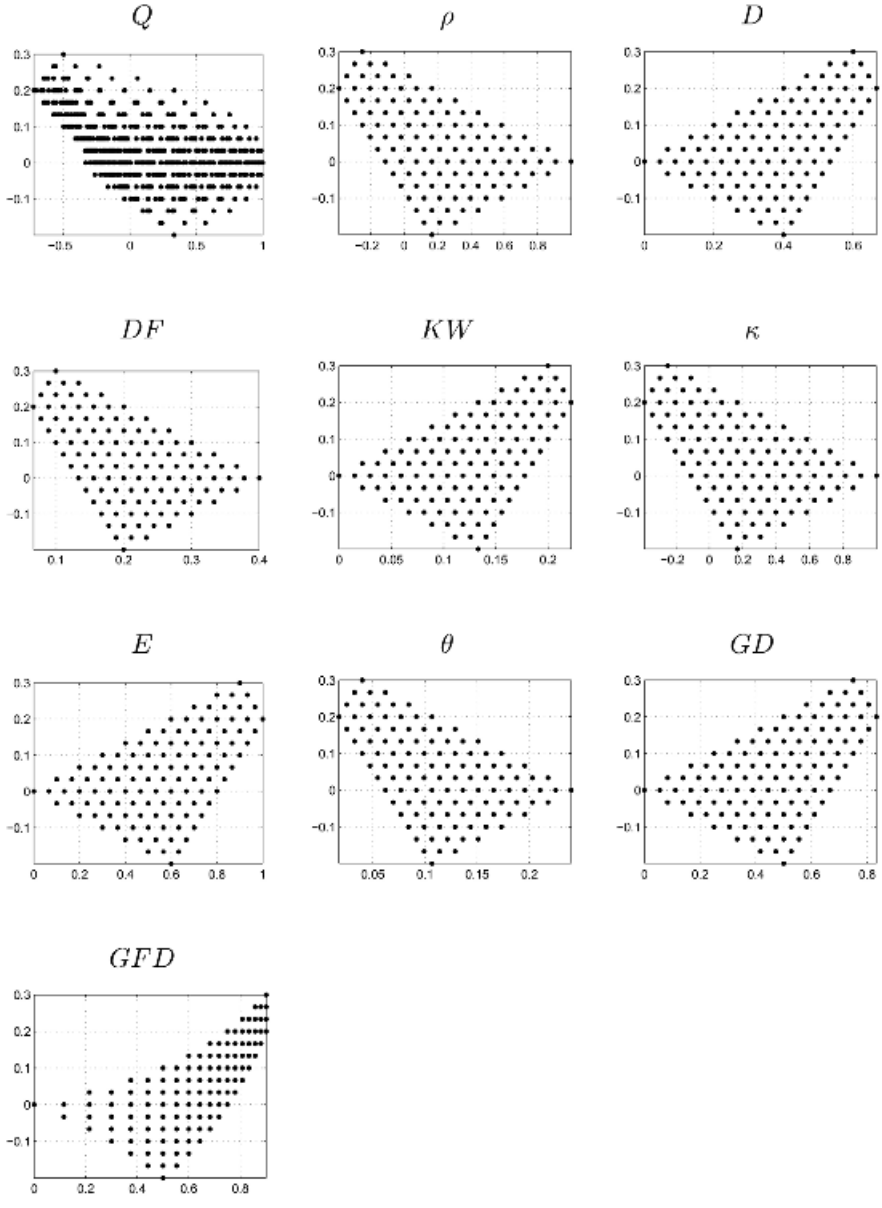


Fig. 10.6 *Improvement on the individual accuracy (P_{maj}-p) versus the 10 diversity measures for $N = 30$, $p = 0.6$.*

So far we assumed that all feasible distributions of the classifier votes are equally likely. Next we assume approximately equal individual accuracies p and approximately equal pairwise dependencies $Q_{i,j}$. We vary $Q_{i,j}$ (which in this case is the same as Q_{av}) from -1 to 1. Classifier *outputs* were generated in the form of correct/incorrect votes. Within the generation algorithm adopted [302], it was not always possible to maintain the values of $Q_{i,j}$ equal for all the pairs.

Six sets of experiments were carried out with the individual accuracy $p \in \{0.6, 0.7\}$ and the number of classifiers $L \in \{3, 5, 9\}$. In all experiments $N = 1000$. For each of the six combinations of p and L, 15 classifier ensembles $\mathcal{D}$ were generated for each of the 21 values of the averaged pairwise dependency $Q_{\text{av}} \in \{-1.0, -0.9, \ldots, 0.9, 1.0\}$, giving a total of 315 classifier ensembles.

In summary,

1. All individual classifiers have approximately the same accuracy (prespecified).
2. The pairwise dependency was approximately the same (prespecified).
3. Not all negative values of Q_{av} were possible for all L. This means that the distribution of Q_{av}, which was intended to range uniformly from -1 to $+1$ spanned a shorter range from $-a$ to 1, where $a \in (0, 1)$.

Let P_{mean} and P_{max} be the observed mean and maximum accuracies, respectively, of the generated ensemble $\mathcal{D}$. For each combination of L and p, we calculated the correlation between each of $P_{\text{maj}} - P_{\text{mean}}$ and $P_{\text{maj}} - P_{\text{max}}$ with the 10 measures. All measures exhibited high (by absolute value) correlation as summarized in Table 10.6.

The relationship between Q_{av} and $P_{\text{maj}} - P_{\text{max}}$ is illustrated graphically in Figure 10.7. Each point in the scatterplot corresponds to a classifier ensemble. Smaller Q (more diverse classifiers) leads to higher improvement over the single best classifier. Negative Q (negative dependency) is better than independence ($Q = 0$) as it leads to an even bigger improvement. The zero improvement is marked with a horizontal line in Figure 10.7. The points below the line correspond to

TABLE 10.6 Extreme Values of the Rank Correlation Coefficients Between Measures of Diversity and the Improvement on the Single Best Accuracy and the Mean Accuracy of the Ensemble.

	$P_{\text{maj}} - P_{\text{max}}$		$P_{\text{maj}} - P_{\text{mean}}$	
	$p = 0.6$	$p = 0.7$	$p = 0.6$	$p = 0.7$
Minimum	0.9371	0.9726	0.9652	0.9826
Maximum	0.9870	0.9923	0.9909	0.9949

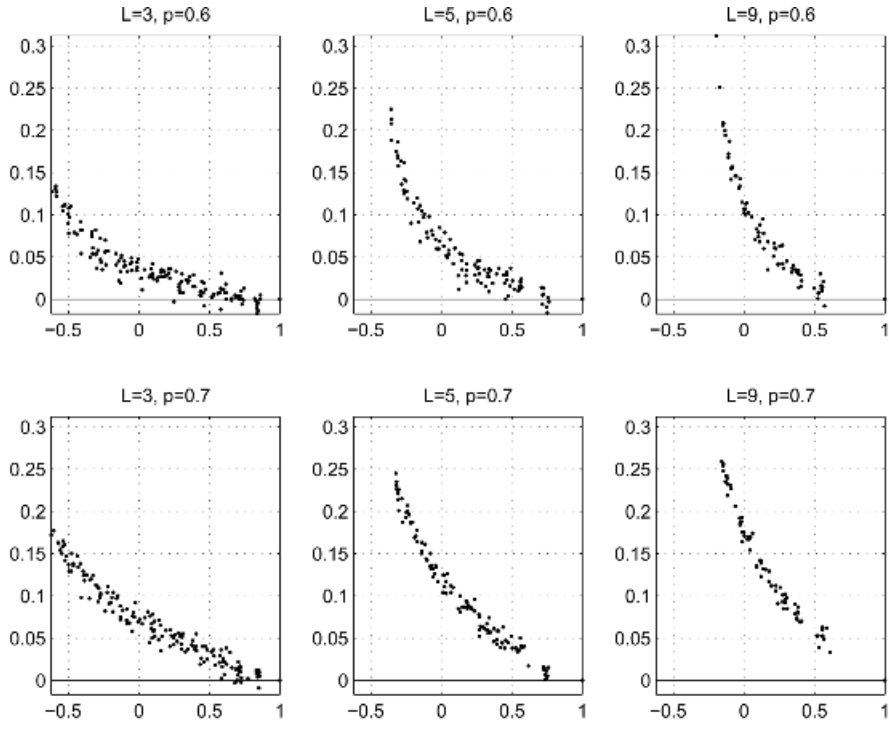


Fig. 10.7 *Plot of* $(P_{maj} - P_{max})$ *versus* Q_{av} *for the simulation experiment.*

classifier ensembles that fared worse than the single best classifier. In all these cases, the ensembles consist of positively related but not identical classifiers.

Although the suspected relationship appears on a large scale, that is, when diversity spans (uniformly) the whole range of possible values, in practice we are faced with a different picture. Usually the candidates for the ensemble are not very different from one another. This leads to small variations of diversity and also small variations of the accuracy of the ensemble about the individual accuracies. Unfortunately, none of the various diversity measures that we investigated previously (10 measures: 4 pairwise and 6 nonpairwise [303]) appeared to be sensitive enough to detect the changes in the accuracy. This phenomenon is illustrated in Figure 10.8 showing a typical graph of accuracy versus diversity. The relationship can easily be spotted on the plot. However, when diversity only varies in a small range, this relationship is blurred as illustrated by the gray dot and the cloud of classifiers in it.

If we do not enforce diversity, the ensemble is most likely to appear as a dot towards the right side of the graph. For such ensembles, the improvement on the individually best accuracy is usually negligible.

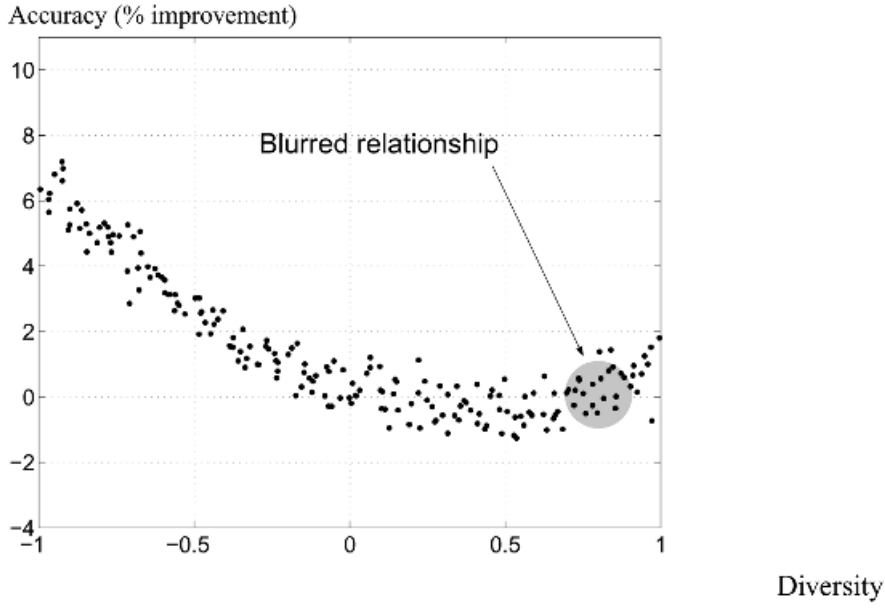


Fig. 10.8 *A typical accuracy–diversity scatterplot. Each point corresponds to an ensemble. The gray dot shows a hypothetical area where ensembles appear most often in real problems.*

10.4 USING DIVERSITY

10.4.1 Diversity for Finding Bounds and Theoretical Relationships

Assume that classifier outputs are estimates of the *posterior probabilities*, $\hat{P}_i(\omega_s|\mathbf{x})$, $s = 1, \ldots, c$, $i = 1, \ldots, L$, so that the estimate $\hat{P}_i(\omega_s|\mathbf{x})$ satisfies

$$\hat{P}_i(\omega_s|\mathbf{x}) = P(\omega_s|\mathbf{x}) + \eta_s^i(\mathbf{x}) \tag{10.37}$$

where $\eta_s^i(\mathbf{x})$ is the error for class ω_s made by classifier D_i. The outputs for each class are combined by averaging, or by an order statistic such as minimum, maximum, or median. In Chapter 9 we derived an expression about the added classification error (i.e., the error above the Bayes error) of the ensemble under a set of assumptions

$$E_{\text{add}}^{\text{ave}} = E_{\text{add}}\left(\frac{1 + \delta(L-1)}{L}\right) \tag{10.38}$$

where E_{add} is the added error of the individual classifiers (all have the same error), and δ is a correlation coefficient (the measure of diversity of the ensemble).[42]

[42] Averaged pairwise correlations between $P_i(\omega_s|\mathbf{x})$ and $P_j(\omega_s|\mathbf{x})$, $i, j = 1, \ldots, L$, are calculated for every s, then weighted by the prior probabilities $\hat{P}(\omega_s)$ and summed.

Breiman [214] derives an upper bound on the generalization error of random forests (ensembles of decision trees built according to a simple randomization technology, one possible variant of which is bootstrap sampling) using the averaged pairwise correlation between the ensemble members. The classifiers produce class labels, which are combined by the majority vote. The bound is given by

$$Pr(\text{generalization error of the ensemble}) \quad \leq \quad \bar{\rho}(1-s^2)s^2 \tag{10.39}$$

where $\bar{\rho}$ is the averaged pairwise correlation (10.6) and s is the "strength" of the ensemble. The strength is a measure of accuracy based on the concept of margin. Admittedly the bound is not very tight as it is based on the Chebyshev's inequality but nonetheless it shows the tendency: the higher the diversity (small $\bar{\rho}$), the lower the error.

10.4.2 Diversity for Visualization

Diversity measures have been used to find out what is happening within the ensemble. Pękalska and co-authors [304] look at a two-dimensional plot derived from the matrix of pairwise diversity. Each classifier is plotted as a dot in the two-dimensional space found by Sammon mapping, which preserves the distances between the objects. Each point in the plot represents a classifier and the distances correspond to pairwise diversities. The ensemble is a classifier itself and can also be plotted. Any method of combination of the individual outputs can also be mapped. Even more, the oracle classifier (all objects correctly recognized) can be plotted as a point to complete the picture.

Margineantu and Dietterich suggest the kappa–error plots as shown in Figure 10.9 [305]. Every pair of classifiers is plotted as a dot in a two-dimensional space. The pairwise measure kappa (10.8) is used as the x-coordinate of the point and the average of the individual training errors of the two classifiers is used as the y-coordinate. Thus there are $L(L-1)/2$ points in the scatterplot. The best pairs are situated in the left bottom part of the plot: they have low error and low kappa (low agreement = high diversity).

The cloud of points shows the pairwise diversity in one ensemble. Margineantu and Dietterich use it to verify that AdaBoost generates more diverse classifiers than bagging. The example in the figure corresponds to an ensemble of 50 classifiers for the *glass* data set from the UCI Machine Repository Database.[43] The shape of the cloud indicates that there is a certain trade-off between the accuracy of the pair and its κ-diversity.

10.4.3 Overproduce and Select

Several studies advocate the method of producing a pool of classifiers followed by a selection procedure to pick the classifiers that are most diverse and accurate [202,238,299,300,305–308]. Below we explain some of the ideas.

[43] http://www.ics.uci.edu/~mlearn/MLRepository.html

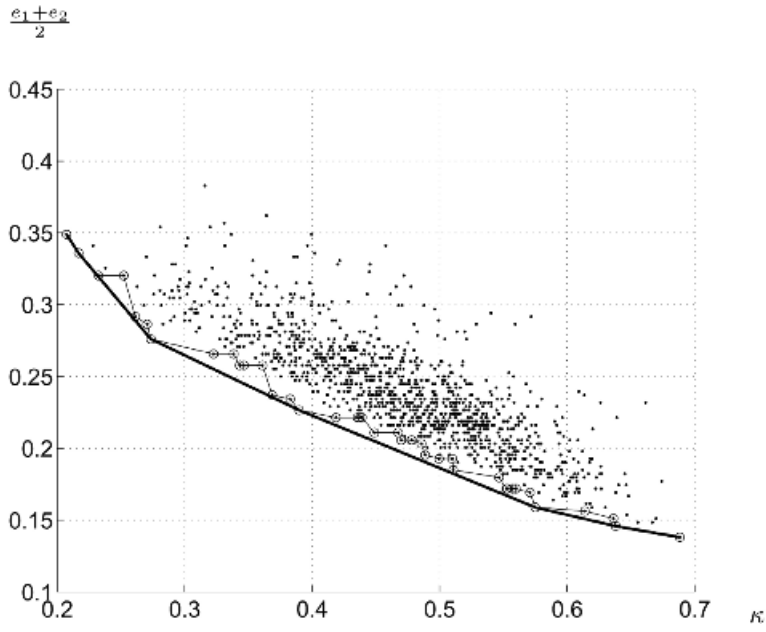


Fig. 10.9 *Kappa–error plot, the convex hull (the thick line), and the Pareto optimal set (the thin line) of pairs of classifiers.*

10.4.3.1 The Most Diverse Ensemble. Giacinto and Roli [202] use the *double fault* measure (probability of both classifiers being incorrect, $DF = d$ in Eq. (10.13)) and also the Q statistics [307], to form a pairwise diversity matrix for a classifier pool and subsequently to select classifiers that are least related. The selection is carried out using a search method through the set of pairs of classifiers until the desired number of ensemble members is reached. Similarly, Margineantu and Dietterich [238,305] use kappa to select the ensemble out of the set of classifiers produced by AdaBoost. They call this "ensemble pruning." The pruned ensemble is created by progressively selecting pairs with lowest kappas (highest diversity) until the desired number of classifiers is reached. A Matlab function for selecting the "most diverse ensemble" is given in Appendix 10B.

10.4.3.2 Clustering and Selection. Giacinto and Roli [306] cluster the ensembles based on the matrix of pairwise diversity. The double fault measure of diversity (DF) is used. The matrix with diversities is regarded as a *distance* matrix and the average linkage clustering is applied (see Section 1.7) to find clusters of classifiers. Grouped in this way, the members of each cluster tend to make common errors while two classifiers selected from different clusters will tend to make errors on different objects. At each step of the procedure, one classifier is taken from each cluster, for example, the classifier with the highest individual accuracy, to be

a member of the ensemble. Thus at the first step where each classifier is a cluster of its own, there are L clusters, hence the ensemble consists of all classifiers. At the next step, the two least diverse classifiers are joined in a cluster so there are $L-1$ members of the ensemble. The classifier picked from the two members of the cluster will be the more accurate of the two. Next the distances to the new cluster are recalculated by the average method and the next two least diverse clusters are merged. The procedure goes through L steps, producing ensembles of L, $L-1, \ldots, 2$, and 1 classifiers. At the first step there is no selection and at the last step the single best classifier is chosen instead of the ensemble. Giacinto and Roli suggest to find the optimal ensemble size L^* to be the size of the most accurate ensemble. To reduce the optimistic bias of the estimates, the accuracy of the ensemble should be evaluated on a validation set that is different from the set on which the classifiers were trained. A Matlab function implementing the clustering and selection method is given in Appendix 10B. It requires a clustering function based on a distance matrix, which in our case is the matrix of pairwise diversities. Examples of suitable clustering procedures are single linkage and average linkage.

10.4.3.3 Thinning the Ensemble. Banfield et al. [300] propose an algorithm for what they call "thinning the ensemble." They introduce a diversity measure called the *ensemble diversity measure* (EDM), which is calculated as the proportion of data points for which the proportion of correct votes is between 10 and 90. We can refer to these points as "uncertain points." The uncertain points will vary depending on which classifiers are chosen for the ensemble. The idea of the thinning algorithm is to find out which classifier is most often incorrect on the uncertainty points and remove it from the ensemble. The procedure starts with all classifiers and the desired ensemble size is reached by removing one classifier at each step. The original thinning algorithm is shown below:

1. Start with the ensemble of all L classifiers.
2. Set the value of the parameter α (recommended $\alpha = 0.9$).
3. Find out the uncertainty points in the data set, that is, the points for which the proportion of correct votes is between 0.1 and 0.9 (uncertainty points).
4. Calculate the diversity (EDM) as the proportion of uncertainty points.
5. Calculate the mean classifier accuracy, m, for the members of the current ensemble.
6. Calculate the lower bound (LB) and the upper bound (UB) needed for the thinning

$$LB = m \times EDM + \frac{1 - EDM}{c} \quad \text{and} \quad UB = \alpha \times EDM + m \times (1 - EDM)$$

   where c is the number of classes.
7. Find the classifier with the lowest accuracy on all the points whose proportion of correct votes is between LB and UB. (Note that this is a different set of uncertain points; the bounds are used in the place of the constants 0.1 and 0.9.)

8. Remove this classifier and continue with the new ensemble from Step 3 until the desired ensemble size is reached.

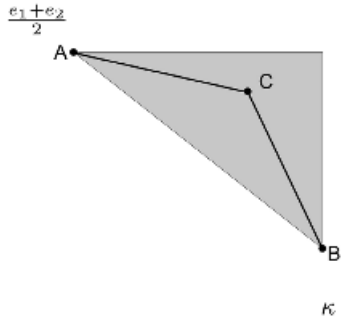
The bounds vary depending on the diversity of the current ensemble.

It seems that the thinning might succeed even without considering *EDM*. We can just apply the main idea of removing the least accurate classifier for the uncertainty points found at Step 3. We can skip Steps 2, 4, 5, and 6, and use at Step 7 the set of uncertainty points found at Step 3, that is, fix $LB = 0.1$ and $UB = 0.9$. The code for this variant of the thinning method is given in Appendix 10B.

10.4.3.4 Kappa–Error Convex Hull Pruning. Margineantu and Dietterich [238,305], use the kappa–error plots. As the most desirable pairs of classifiers are situated toward the lower left corner of the plot, Margineantu and Dietterich use the convex hull [305], called *kappa–error convex hull pruning*. The convex hull of points is depicted in Figure 10.9 with a thick line. The pairs of classifiers that make the convex hull are taken as the ensemble.

10.4.3.5 Pareto Optimality. It might happen that the convex hull contains only a few classifiers on the frontier. Small variations of the estimates of κ and $\frac{e_1+e_2}{2}$ might change the whole frontier, making convex hull pruning overly sensitive to noise. The number of classifiers in the pruned ensemble cannot be specified in advance. This lack of control on the ensemble size is seen as a defect of the method [305].

Therefore we may look at *Pareto optimality* as an alternative to the convex hull approach. Let $A = \{a_1, \ldots, a_m\}$ be a set of alternatives (pairs in our case) characterized by a set of criteria $C = \{C_1, \ldots, C_M\}$ (low kappa and low error in our case). The Pareto-optimal set $S^* \subseteq S$ contains all nondominated alternatives. An alternative a_i is non-dominated iff there is no other alternative $a_j \in S$, $j \neq i$, so that a_j is better than a_i on *all* criteria. For the two criteria in our example, the Pareto optimal set will be a superset of the convex hull. The concept is illustrated in Figure 10.10. The Pareto-optimal set for the glass data example is depicted in Figure 10.9 by a thin line joining the circled points in the set.



Suppose that points A and B are in the convex hull. Point C is not in the convex hull because it is "behind" the segment AB. However, C is better than A on the error criterion and better than B on the kappa criterion. Therefore C is nondominated, so it belongs in the Pareto optimal set.

Fig. 10.10 *Illustration of Pareto optimality.*

A Matlab function for finding the Pareto-optimal set of alternatives on two criteria is given in Appendix 10B. In our case the alternatives are the pairs of classifiers and the two criteria for each pair are the averaged individual error and the diversity. The code is designed in such a way that high values are more desirable than low values. Therefore, the function should be called using $-(e_i + e_j)/2$ and $-\kappa_{i,j}$ as the input.

Example: Illustration of Overproduce and Select Methods. Recall the rotated check-board data from Chapter 1. We used a data set generated from the distribution illustrated in the right-hand side plot of Figure 1.10 ($a = 0.5$, $\alpha = -\pi/3$). There are two classes that are difficult to separate using a simple classifier, but which are nonoverlapping. We generated 100 points for training and 1000 points for testing. Ideally there should be a separate validation set. For the purposes of this simple illustration, however, we will use the training set both for training the classifiers, estimating their accuracies and finding the optimal size of the ensemble.

Figure 10.11 shows the training and testing accuracy of the ensemble when adding classifiers designed through bagging. The kappa–error plot for the final ensemble of $L = 50$ classifiers is shown in Figure 10.12, where the pairs of classifiers on the convex hull of the cloud of points are joined by a thick line and the pairs of classifiers in the Pareto-optimal set are circled and joined by a thin line.

The methods and algorithms discussed above were applied for this single run of bagging using $\kappa_{i,j}$ as the diversity measure. The convex hull selection and Pareto-

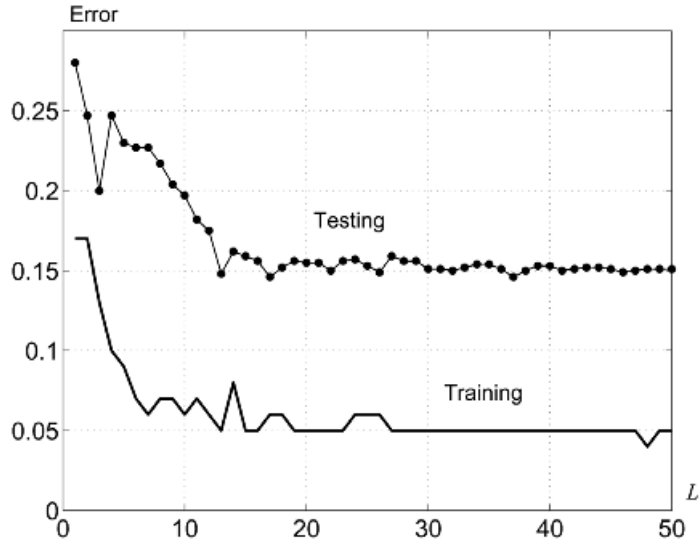


Fig. 10.11 *Training and testing error for one run of bagging for the rotated checkboard data versus the number of classifiers.*

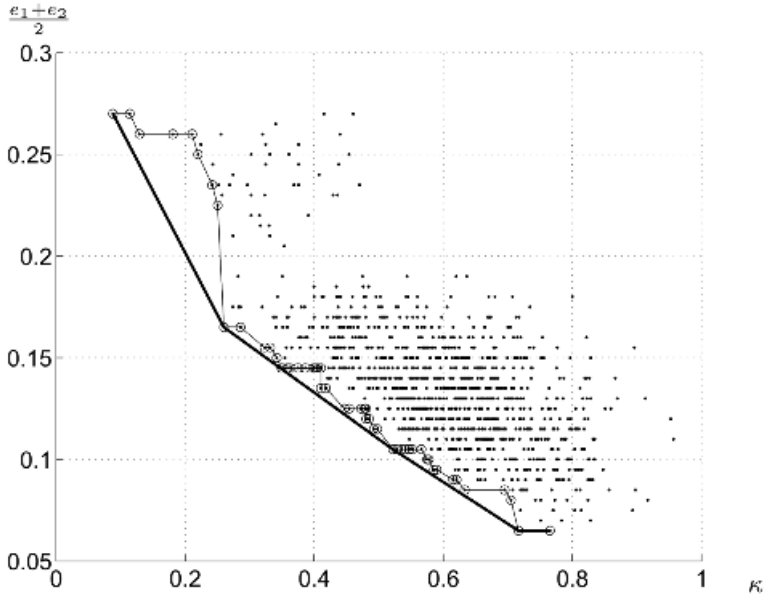


Fig. 10.12 *Kappa–error plot for 50 classifiers obtained by bagging for the rotated check-board data. The Pareto-optimal set and the convex hull set of classifier pairs are shown.*

optimal selection do not have a mechanism to control the number of classifiers in the ensemble. The convex hull will always result in ensembles of smaller or equal size to these found by Pareto-optimal selection. The reason for this is that a Pareto-optimal set always contains the convex hull as a subset. All the other methods can be run for ensembles of sizes from 1 to L. The optimal ensemble size, L^*, can be identified as the size of the most accurate ensemble evaluated on a validation set. In our case we used the training set to find L^*. Figure 10.13 shows the testing errors of the overproduce and select methods versus the ensemble size. The convex hull and Pareto-optimal selections are depicted as points with coordinates (size, error). The method based on selecting the *most diverse* ensemble is the worst of the group as it does not take into account the performance of the classifiers or the ensemble. The best selection method for a small number of classifiers happened to be the *ensemble thinning* version. Convex hull gave also an excellent result, but such good results have not been observed on a large experimental scale [305]. *Clustering and selection* was marginally worse than the thinning method. The original thinning algorithm was also applied, but the results were not much different to those of the simplified version.

Table 10.7 shows the ensemble sizes L^* identified on the training sets and the corresponding testing errors. The example shows that smaller ensembles can be as good as or sometimes better than larger ensembles.

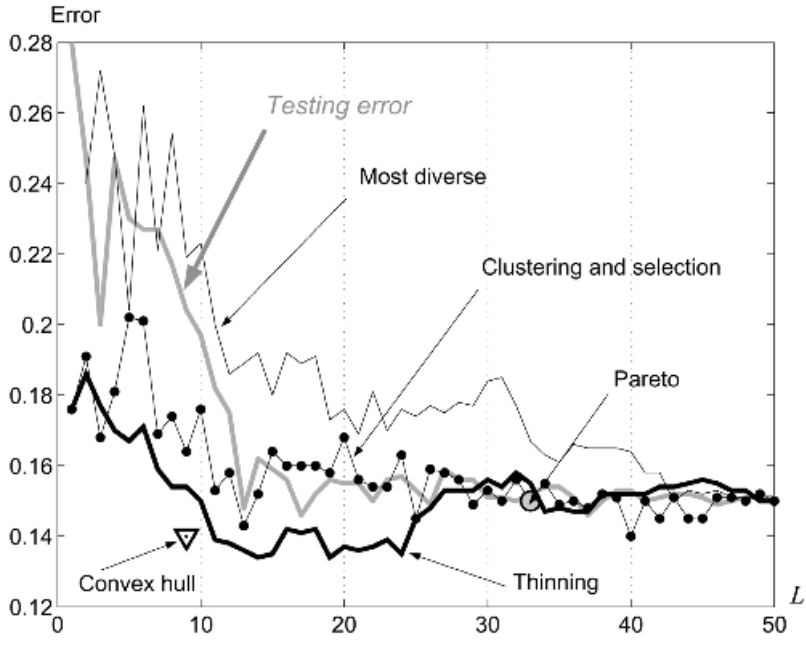


Fig. 10.13 *Testing error versus the number of classifiers for the overproduce and select methods.*

Classifier selection has been found not as efficient in improving the accuracy on AdaBoost. Usually the most accurate ensemble is the one with all the classifiers in it [305]. The reason for this could be that AdaBoost explores diversity by design, and does not create redundant members of the ensemble.

Instead of forming clusters and selecting one classifier from each cluster or resort to a greedy algorithm that may not perform particularly well, some authors consider using genetic algorithms for selecting the ensemble [309,310]. Each chromosome has L bits and corresponds to an ensemble. The bits for the classifiers that participate in the ensemble are set to 1, and the remaining bits are set to 0. Each population contains ensembles as the individuals. Using genetic algorithms for classifier selection

TABLE 10.7 Testing Error and Ensemble Sizes for the Overproduce and Select Methods for the Rotated Check-Board Data Set.

Method	Testing Error [%]	Ensemble Size (L^*)
The whole ensemble	15.1	50
The most diverse ensemble	16.9	21
Clustering and selection	15.2	14
Thinning the ensemble	15.9	7
Kappa–error convex hull pruning	14.0	9
Pareto-optimality	15.0	33

pays off when the search space is large and the calculation of the criterion function (ensemble accuracy, ensemble diversity, individual accuracy, and so on) is cheap.

10.4.4 Diversity for Building the Ensemble

Here we only sketch a possible way of using diversity explicitly during the *process of building of the ensemble*. The motivation is that diversity should step out of the passive role of being only a tool for monitoring and should help actively at the design stage. The overproduce and select approach discussed earlier is a step in this direction. However, we need to overproduce first. An alternative approach would be to stop the growing of the ensemble when diversity and accuracy satisfy a certain condition.

We take as the starting point the kappa–error diagram and run AdaBoost.[44] The first and the second classifier (D_1 and D_2) will define one single point on the diagram. This point will be the convex hull and the Pareto optimal set of itself. The third classifier, D_3, will place two more points: one for (D_1, D_3) and another for (D_2, D_3). At this step we recalculate the Pareto-optimal set. If the points added by the new classifier have not changed the previous Pareto-optimal set, then this classifier is not accepted. Another training set is generated with the same distribution and a new classifier is attempted on it. We run the acceptance check again, and proceed in this manner. A prespecified parameter T defines the limit of the number of attempts from the same distribution. When T attempts have been made and a classifier has not been accepted, the procedure stops and the classifier pairs in the last Pareto optimal set are declared to be the ensemble.

10.5 CONCLUSIONS: DIVERSITY OF DIVERSITY

Although diversity is perceived as one of the most important characteristics of the ensemble, there is still a diversity of opinions about it. We exploit diversity at an intuitive level in creating the ensemble, but so far it has defied the attempts to be measured and used systematically. The problem might be in the philosophy of the concept of diversity. Three views on diversity are detailed below.

1. *Diversity as a characteristic of the set of classifiers.* We have a set of classifiers and we have not decided yet which combiner to use. Also, we do not involve information about whether or not the classifier votes are correct. This view seems to be the "cleanest." It would provide information additional to the individual error rates and the ensemble error rate. In a way, we measure diversity to *discover* whether it contributes to the success of the ensemble.
2. *Diversity as a characteristic of the set of classifiers and the combiner.* In this case the ensemble output is also available. Thus we can find out which classifier deviates the most and which deviates the least from the ensemble output,

[44] We use AdaBoost in its resampling version: the likelihood of data points to be selected is modified.

and measure diversity on such an individual basis. Different combiners might lead to different diversity values for the same set of classifiers.

3. *Diversity as a characteristic of the set of classifiers, the combiner, and the errors.* Here we can also use the oracle information available. Diversity is seen as a component of the ensemble error and formulas have been sought to relate diversity to the ability of the ensemble to improve on the individual classifier performances. Here we are actively searching for a useful relationship between diversity and the ensemble error that will guide us toward construction of better ensembles [292,301].

The problem is that the "clean" diversity measure might be of little use due to its weak relationship with the ensemble accuracy [303,311]. On the other hand, the more we involve the ensemble performance into defining diversity, the more we are running onto the risk of trying to replace a simple calculation of the ensemble error by a clumsy estimate that we call diversity. Interpretability of the measure as *diversity* might be lost on the way of trying to tie it up with the ensemble error.

There is no easy solution to this dilemma. The quest for defining and using diversity might be heading toward a dead end or might result in powerful new ensemble-building methodologies. In science we do not know the answer in advance and this is what makes it fun.

APPENDIX 10A EQUIVALENCE BETWEEN THE AVERAGED DISAGREEMENT MEASURE D_{av} AND KOHAVI–WOLPERT VARIANCE *KW*

Recall that $Y(\mathbf{z}_j)$ is the number of correct votes (1s) for object $\mathbf{z}_j$. The Kohavi–Wolpert variance [231], in the case of two alternatives, 0 and 1, is

$$KW = \frac{1}{NL^2}\sum_{j=1}^{N} Y(\mathbf{z}_j)(L - Y(\mathbf{z}_j)) \tag{A.1}$$

$$= \frac{1}{NL^2}\sum_{j=1}^{N}\left(\sum_{i=1}^{L} y_{j,\,i}\right)\left(L - \sum_{i=1}^{L} y_{j,\,i}\right) = \frac{1}{NL^2}\sum_{j=1}^{N}\mathcal{A}_j \tag{A.2}$$

where

$$\mathcal{A}_j = \left(\sum_{i=1}^{L} y_{j,\,i}\right)\left(L - \sum_{i=1}^{L} y_{j,\,i}\right) \tag{A.3}$$

The disagreement measure between D_i and D_k used in Ref. [295] can be written as

$$D_{i,k} = \frac{1}{N}\sum_{j=1}^{N}(y_{j,i} - y_{j,k})^2 \tag{A.4}$$

Averaging over all pairs of classifiers i, k,

$$D_{av} = \frac{1}{L(L-1)} \sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} \frac{1}{N} \sum_{j=1}^{N} (y_{j,i} - y_{j,k})^2 \tag{A.5}$$

$$= \frac{1}{NL(L-1)} \sum_{j=1}^{N} \sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} (y_{j,i} - y_{j,k})^2$$

$$= \frac{1}{NL(L-1)} \sum_{j=1}^{N} \mathcal{B}_j \tag{A.6}$$

where

$$\mathcal{B}_j = \sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} (y_{j,i} - y_{j,k})^2 \tag{A.7}$$

Dropping the index j for convenience and noticing that $y_i^2 = y_i$

$$\mathcal{A} = L\left(\sum_{i=1}^{L} y_i\right) - \left(\sum_{i=1}^{L} y_i\right)^2 \tag{A.8}$$

$$= L\left(\sum_{i=1}^{L} y_i\right) - \left(\sum_{i=1}^{L} y_i^2\right) - \left(\sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} y_i y_k\right) \tag{A.9}$$

$$= (L-1)\left(\sum_{i=1}^{L} y_i\right) - \left(\sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} y_i y_k\right) \tag{A.10}$$

On the other hand,

$$\mathcal{B} = \sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} (y_i^2 - 2y_i y_k + y_k^2) \tag{A.11}$$

$$= 2(L-1)\left(\sum_{i=1}^{L} y_i\right) - 2\left(\sum_{i=1}^{L} \sum_{k=1_{i \neq k}}^{L} y_i y_k\right) \tag{A.12}$$

$$= 2\mathcal{A} \tag{A.13}$$

Therefore,

$$KW = \frac{L-1}{2L} D_{\text{av}} \tag{A.14}$$

Since the two diversity measures differ by a coefficient, their correlation with $P_{\text{maj}} - P_{\text{mean}}$ will be the same.

APPENDIX 10B MATLAB CODE FOR SOME OVERPRODUCE AND SELECT ALGORITHMS

The *most diverse ensemble* uses a matrix of pairwise diversities, D. The higher the value, the better the diversity. The output variable "ens" contains the indices of the classifiers that make up the ensemble. The maximum value of diversity is found, the classifiers added to the ensemble and then the maximum value is set to $-\infty$ so that it is not found again. It may happen that none of the classifiers in the pair with the (current) highest diversity is in the selected set, one of the pair is already in the set or both are in the set. Thus at each step either one of two classifiers may enter the selected set. The function returns the indices of the selected classifiers.

```
function ens = most_diverse_ensemble(D,M)
%
% Returns in ens the list of classifier indices
% which make the ensemble of M or M+1 classifiers
% according to the greedy algorithm called "most
% diverse ensemble" described in the text.
% D is a matrix with pairwise diversity.
% M is the desired ensemble size
% We assume that larger values of D indicate
% high diversity
%
L=size(D,1); % Find out the total number of classifiers
mask=zeros(1,L); % Start with an empty ensemble
while sum(mask)<M,
    % Find the pair with the highest diversity
    [a1,a2]=max(D);
    [b1,b2]=max(a1);
    % The classifier pair with largest
    % diversity is (a2(b2),b2)
    mask(a2(b2))=1;
    mask(b2)=1;
    D(a2(b2),b2)=-inf; % Disable the current maximum
end;
ens=find(mask); % Assign the classifier indices to
                % the output variable
```

The following function implements the *clustering and selection* method. It requires a clustering function operating on a distance matrix. The distance matrix in our case is the matrix of pairwise diversities D. The clustering function 'cluster' must output a vector column of numerical class labels for the N objects. The cluster labels should be consecutive integers: 1,2, . . . , M.

```
function ens = clustering_and_ensemble_selection (l,M,laba,D)
%
% l is the array of size N × L containing
% the outputs of the L classifiers on a data set
% of N objects (preferably a validation set)
% The classes should be labeled by consecutive
% integers: 1,2,...,c
% M is the desired ensemble size
% laba is a vector-column with the true labels
% of the N objects
% D is a matrix with pairwise diversity.
%
[N,L]=size(l);
p=mean(l== repmat(laba,1,L)); % Estimate individual accuracies
labels=cluster(D,M); % Cluster the classifiers
% Find the best individual classifier in each cluster
mask=zeros(1,L); % Start with an empty ensemble
for j=1:M,
    t1=find(labels== j); % Find all classifiers in cluster j
    [t2,t3]=max(p(t1));
    % t1(t3) is the most accurate classifier
    mask(t1(t3))=1; % Add it to the ensemble
end;
ens=find(mask);
```

The version of the *thinning the ensemble* algorithm described in the text is given as a Matlab function below.

```
function ens = thinning_the_ensemble(l,M,laba)
%
% A simpler variant of the ensemble thinning method [18]
% l is the array of size N × L containing
% the outputs of the L classifiers on a data set
% of N objects (preferably a validation set)
% The classes should be labeled by consecutive
% integers: 1,2,...,c
% M is the desired ensemble size
% laba is a vector-column with the true labels
% of the N objects
%
[N,L]=size(l);
mask=ones(1,L);% Start with the whole ensemble
oracle=(l== repmat(laba,1,L)); % Create a binary
     % matrix with the correct/incorrect outputs
```

```
for i=L:-1:2, % Remove one classifier at a time
    % Using the current ensemble, find the
    % ''uncertain'' points
    uncertain_points=(mean(oracle (:,find (mask))')>0.1)...
                    & (mean(oracle(:,find(mask))')<0.9);
    % Calculate a vector, p, with the accuracies
    % of the L classifiers
    uncertain_labels=repmat(laba(find(uncertain_points)),1,L);
    p=mean(l(find(uncertain_points),:)==uncertain_labels);
    % Eliminate the classifiers that have been discarded
    p(find(mask))=inf;
    [t1,t2]=min(p); % Find the least accurate classifier
    % for the uncertain points (t2)
    mask(t2)=0; % Remove classifier t2
end;
ens=find(mask);
```

The following function finds the *Pareto-optimal* set (the set of nondominated alternatives) on two criteria for a set of N alternatives. The alternatives correspond to the rows of X; each row contains the two criteria values for the respec-tive alternative. The code outputs the Pareto-optimal set (a matrix with the respective rows from X) and the indices of the rows of X that constitute the Pareto-optimal set.

```
function [P,set_of_indices]=Pareto(X);
%
% Calculates the Pareto optimal subset P of an array X (Nx2)
% Assumes "the higher the better" on both coordinates
%
set_of_indices=[1];
if size(X,1)==1, % If X contains only one row,
    P=X; % then this is also the Pareto optimal set
else
    P=min(X); % Initialize the Pareto optimal set
    for i=1:size(X,1),
        index=(P(:,1)>X(i,1))&(P(:,2)>X(i,2));
        if sum(index)==0, % X(i,:) is non-dominated
            % Calculate the elimination index
            % to find out which alternatives in the set
            % are dominated by X(i,:)
            index_el=(P(:,1)<X(i,1))&(P(:,2)<X(i,2));
            P=P(~index_el,:); % Update P
            P=[P;X(i,:)]; % Add X(i,:)
            set_of_indices=[set_of_indices;i];
        end;
    end;
end;
```

References

1. P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.
2. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, N.Y., 1973.
3. K.-S. Fu. *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, N.J, 1982.
4. K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Inc., Orlando, FL, 1972.
5. M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Mass, 1969.
6. N. J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1962.
7. E. A. Patrick. *Fundamentals of Pattern Recognition*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1972.
8. F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington D.C., 1962.
9. G. S. Sebestyen. *Decision-Making Process in Pattern Recognition*. The Macmillan Company, N.Y., 1962.
10. J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, Reading, MA, 1974.
11. J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.
12. N. Glick. Additive estimators for probabilities of correct classification. *Pattern Recognition*, 10:211–222, 1978.

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1

13. G. T. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, 20:472–479, 1974.
14. T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 7(10):1895–1924, 1998.
15. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*, 2nd ed. John Wiley & Sons, NY, 2001.
16. R. P. W. Duin. A note on comparing classifiers. *Pattern Recognition Letters*, 17:529–536, 1996.
17. A. K. Jain and J. Mao. Guest editorial: Special issue on artificial neural networks and statistical pattern recognition. *IEEE Transactions on Neural Networks*, 8(1):1–3, 1997.
18. L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.
19. R. P. W. Duin. PRTOOLS(Version 2). A Matlab toolbox for pattern recognition. Pattern Recognition Group, Delft University of Technology, June 1997.
20. J. L. Fleiss. *Statistical Methods for Rates and Proportions*. John Wiley & Sons, 1981.
21. P. Latinne, O. Debeir, and C. Decaestecker. Different ways of weakening decision trees and their impact on classification accuracy of DT combination. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000. Springer, pp. 200–209.
22. S. W. Looney. A statistical technique for comparing the accuracies of several classifiers. *Pattern Recognition Letters*, 8:5–9, 1988.
23. E. Alpaydin. Comparison of statistical and neural classifiers and their applications to optical character recognition and speech classification. In C. T. Leondes, editor, *Image Processing and Pattern Recognition*, volume 5 of *Neural Network Systems*, Academic Press, 1998, pp. 61–88.
24. G. Nagy. Candide's practical principles of experimental pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):199–200, 1993.
25. S. Ghahramani. *Fundamentals of Probability*, 2nd Ed. Prentice Hall, N.J., 2000.
26. S. G. Alsing, K. W. Bauer, and J. O. Miller. A multinomial selection procedure for evaluating pattern recognition algorithms. *Pattern Recognition*, 35:2397–2412, 2002.
27. C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.
28. C. G. Looney. *Pattern Recognition Using Neural Networks. Theory and Algorithms for Engineers and Scientists*. Oxford University Press, Oxford, 1997.
29. B. D. Ripley. *Pattern Recognition and Neural Networks*. University Press, Cambridge, 1996.
30. H. G. C. Traven. A neural network approach to statistical pattern classification by "semiparametric" estimation of probability density functions. *IEEE Transactions on Neural Networks*, 2(3):366–377, 1991.
31. R. P. Lippmann. A critical overview of neural network pattern classifiers. In *Proc. IEEE Workshop on Neural Networks for Signal Processing*, 1991, pp. 266–275.
32. L. Holmström, P. Koistinen, J. Laaksonen, and E. Oja. Neural and statistical classifiers – taxonomy and two case studies. *IEEE Transactions on Neural Networks*, 8(1):5–17, 1997.
33. P. Arabie, L. J. Hubert, and G. De Soete. *Clustering and Classification*. World Scientific, Singapore, 1996.

34. J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, N.Y, 1981.
35. B. Everitt. *Cluster Analysis*. John Wiley and Sons, N.Y., 1993.
36. A. D. Gordon. *Classification*. Chapman & Hall/CRC, Boca Raton, FL, 1999.
37. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, N.J., 1988.
38. J. H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405):165–175, 1989.
39. L. I. Kuncheva. *Fuzzy Classifier Design*. Studies in Fuzziness and Soft Computing. Springer-Verlag, Heidelberg, 2000.
40. G. A. Babich and O. I. Camps. Weighted Parzen windows for pattern classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):567–570, 1996.
41. R. L. Coultrip and R. H. Granger. Sparse random networks with LTP learning rules approximate Bayes classifiers via Parzen's method. *Neural Networks*, 7:463–476, 1994.
42. M. Pawlak and M. F. Yat Fung Ng. On kernel and radial basis function techniques for classification and function recovering. In *Proc. 12th Int. Conference on Pattern Recognition*, Jerusalem, Israel, 1994, pp. 454–456.
43. D. F. Specht. Probabilistic neural networks. *Neural Networks*, 3(1):109–118, 1990.
44. F. Masulli, F. Casalino, and F. Vannucci. Bayesian properties and performances of adaptive fuzzy systems in pattern recognition problems. In *Proc. European Conference on Artificial Neural Networks, ICANN'94*, Sorrento, Italy, 1994, pp. 189–192.
45. C. -T. Sun and J. -S. Jang. A neuro-fuzzy classifier and its applications. In *Second IEEE International Conference on Fuzzy Systems*, San Francisco, CA, USA, 1993, pp. 94–98.
46. L. X. Wang and J. M. Mendel. Fuzzy basis functions, universal approximation and orthogonal least squares learning. *IEEE Transactions on Neural Networks*, 3(5):807–814, 1992.
47. B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, California, 1990.
48. D. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
49. F. J. Ferri, J. V. Albert, and E. Vidal. Considerations about sample size sensitivity of a family of edited nearest neighbor rules. *IEEE Transactions on Systems, Man, and Cybernetics*, B:29(4):667–672, 1999.
50. L. I. Kuncheva. Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16:809–814, 1995.
51. L. I. Kuncheva. Fitness functions in editing k-nn reference set by genetic algorithms. *Pattern Recognition*, 30:1041–1049, 1997.
52. L. I. Kuncheva and J. C. Bezdek. On prototype selection: Genetic algorithms or random search? *IEEE Transactions on Systems, Man, and Cybernetics*, C28(1):160–164, 1998.
53. V. Cerverón and F. J. Ferri. Another move towards the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 31(3):408–413, 2001.
54. P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 16:515–516, 1968.

55. D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2:408–421, 1972.
56. D. B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proc. 11th Int. Conf. on Machine Learning*, New Brunswick, NJ, 1994, Morgan Kaufmann, Los Altos, CA, pp. 293–301.
57. T. Kohonen. Improved versions of learning vector quantization. In *Proc. International Joint Conference on Neural Networks*, San Diego, CA, June 1990, pp. I545–I550.
58. T. Kohonen. *Self-Organizing Maps*. Springer, Germany, 1995.
59. C. Diamantini and A. Spalvieri. Quantizing for minimum average misclassification risk. *IEEE Transactions on Neural Networks*, 9(1):174–182, 1998.
60. S. Geva and J. Sitte. Adaptive nearest neighbor pattern classification. *IEEE Transactions on Neural Networks*, 2(2):318–322, 1991.
61. R. Odorico. Learning vector quantization with training count (lvqtc). *Neural Networks*, 10(6):1083–1088, 1997.
62. A. Sato and J. Tsukumo. A criterion for training reference vectors and improved vector quantization. In *Proc. IEEE Int. Conference on Neural Networks*, Orlando, FL, 1994, pp. 161–166.
63. Q. Xie, C. A. Laszlo, and R. K. Ward. Vector quantization technique for nonparametric classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1326–1330, 1993.
64. J. C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel. Multiple-prototype classifier design. *IEEE Transactions on Systems, Man, and Cybernetics*, C-28(1):67–79, 1998.
65. C. Decaestecker. Finding prototypes for nearest neighbor classification by means of gradient descent and deterministic annealing. *Pattern Recognition*, 30(2):281–288, 1997.
66. L. Tarassenko and S. Roberts. Supervised and unsupervised learning in radial basis function classifiers. *IEEE Proc.-Vis. Image Signal Process.*, 141(4):210–216, 1994.
67. H. Yan. Handwritten digit recognition using an optimized nearest neighbor classifier. *Pattern Recognition Letters*, 15:207–211, 1994.
68. Y. Hamamoto, S. Uchimura, and S. Tomita. A bootstrap technique for nearest neighbor classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):73–79, 1997.
69. J. C. Bezdek and L. I. Kuncheva. Nearest prototype classifier designs: An experimental study. *International Journal of Intelligent Systems*, 16(12):1445–1473, 2001.
70. T. M. Apostol. *Calculus*, Vol. II, 2nd Ed. Xerox College Publishing, Weltham, Massachusetts, USA, 1969.
71. S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(4):325–327, April 1976.
72. T. Bailey and A. K. Jain. A note on distance-weighted k-nearest neighbor rules. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(4):311–313, 1978.
73. R. L. Morin and D. E. Raeside. A reappraisal of distance-weighted k-nearest neighbor classification for pattern recognition with missing data. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(3):241–243, 1981.

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

75. G. Parthasarathy and B. N. Chatterji. A class of new KNN methods for low sample problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(3):715–718, 1990.

76. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International, Belmont, California, 1984.

77. F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

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

79. A. J. C. Sharkey, editor. *Combining Artificial Neural Nets. Ensemble and Modular Multi-Net Systems*. Springer-Verlag, London, 1999.

80. L. Fausett. *Fundamentals of Neural Networks*. Prentice Hall Inc., Englewood Cliffs, N.J., 1994.

81. M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, Massachusetts, 1995.

82. S. Haykin. *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishing Company, N.Y, 1994.

83. Y.-H. Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, Massachusetts, 1989.

84. D. W. Patterson. *Artificial Neural Networks. Theory and Applications*. Prentice Hall, Simon & Schuster, Singapore, 1996.

85. R. Rojas. *Neural Networks. A Systematic Introduction*. Springer, Berlin, 1995.

86. J. Schürmann. *Pattern Classification. A Unified View of Statistical and Neural Approaches*. John Wiley & Sons, Inc., N.Y., 1996.

87. P. D. Wasserman. *Neural Computing*. Van Nostrand Reinhold, N.Y., 1989.

88. P. D. Wasserman. *Advanced Methods in Neural Computing*. Van Nostrand Reinhold, USA, 1993.

89. J. A. Anderson and E. Rosenfeld. *Neurocomputing. Foundations of Research*. The MIT Press, Cambridge, Massachusetts, 1988.

90. R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, April: 4–22, 1987.

91. R. P. Lippmann. Pattern classification using neural networks. *IEEE Communication Magazine*, November: 47–64, 1989.

92. E. Masson and Y.-J. Wang. Introduction to computation and learning in artificial neural networks. *European Journal on Operational Research*, 47:1–28, 1990.

93. I. Aleksander and H. Morton. *Introduction to Neural Computing*, 2nd ed. Thomson Computer Press, London, 1995.

94. A. Browne, editor. *Neural Network Perspectives on Cognition and Adaptive Robotics*. Institute of Physics Publishing, Bristol, UK, 1997.

95. A. Nigrin. *Neural Networks for Pattern Recognition*. MIT Press, USA, 1993.

96. M. D. Richard and R. P. Lippmann. Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural Computation*, 3:461–483, 1991.

97. D. W. Ruck, S. K. Rojers, M. Kabrisky, M. E. Oxley, and B. W. Suter. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

98. E. A. Wan. Neural network classification: A Bayesian interpretation. *IEEE Transactions on Neural Networks*, 1(4):303–305, 1990.

99. W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

100. K. Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136, 1975.

101. L. Prechelt. PROBEN1 – A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, University of Karlsruhe, Karlsruhe, Germany, 1994.

102. W. Y. Huang and R. P. Lippmann. Comparisons between neural nets and conventional classifiers. In *Proc. IEEE First International Conference on Neural Networks*, San Diego, California, 1987, pp. IV-485–IV-493.

103. I. D. Longstaff and J. F. Gross. A pattern recognition approach to understanding the multi-layer perceptron. *Pattern Recognition Letters*, 5:315–319, 1987.

104. F. Scarselli and A. C. Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1):15–37, 1998.

105. T. K. Ho. Multiple classifier combination: Lessons and the next steps. In A. Kandel and H. Bunke, editors, *Hybrid Methods in Pattern Recognition*. World Scientific Publishing, 2002, pp. 171–198.

106. T. G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000, Springer, pp. 1–15.

107. F. Roli and J. Kittler, editors. *Proc. 1st International Workshop on Multiple Classifier Systems* (*MCS 2000*), Vo. 1857 of *Lecture Notes in Computer Science LNCS* Springer-Verlag, Cambridge, UK, 2001.

108. F. Roli and J. Kittler, editors. *Proc. 2nd International Workshop on Multiple Classifier Systems* (*MCS 2001*), Vol. 2096 of *Lecture Notes in Computer Science LNCS* Springer-Verlag, Cambridge, UK, 2001.

109. F. Roli and J. Kittler, editors. *Proc. 3rd Int. Workshop on Multiple Classifier Systems*(*MCS 2002*), Vol. 2364 of *Lecture Notes in Computer Science LNCS* Springer Verlag, Cagliari, Italy, 2002.

110. T. Windeatt and F. Roli, editors. *Proc. 4th Int. Workshop on Multiple Classifier Systems* (*MCS 2003*), Vol. 2709 of *Lecture Notes in Computer Science LNCS* Springer-Verlag, Guildford, UK, 2003.

111. L. Lam. Classifier combinations: implementations and theoretical issues. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000, Springer, pp. 78–86.

112. G. Valentini and F. Masulli. Ensembles of learning machines. In R. Tagliaferri and M. Marinaro, editors, *Neural Nets, WIRN*, Vol. 2486 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 3–19.

113. R. P. W. Duin. The combining classifier: to train or not to train? In *Proc. 16th International Conference on Pattern Recognition, ICPR'02*, Canada, 2002, pp. 765–770.

114. M. S. Kamel and N. M. Wanas. Data dependence in combining classifiers. In T. Windeatt and F. Roli, editors, *Proc. 4th Int. Workshop on Multiple Classifier Systems* (*MCS 2003*), Vol. 2709 of *Lecture Notes in Computer Science LNCS*, Guildford, UK, 2003, Springer-Verlag, pp. 1–14.

115. C. Dietrich, G. Palm, and F. Schwenker. Decision templates for the classification of bioacoustic time series. *Information Fusion*, 4:101–109, 2003.

116. L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

117. D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

118. B. V. Dasarathy and B. V. Sheela. A composite classifier system design: concepts and methodology. *Proceedings of IEEE*, 67:708–713, 1978.

119. L. A. Rastrigin and R. H. Erenstein. *Method of Collective Recognition*. Energoizdat, Moscow, 1981 (in Russian).

120. K. Woods, W. P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:405–410, 1997.

121. Y. L. Barabash. *Collective Statistical Decisions in Recognition*. Radio i Sviaz', Moscow, 1983 (In Russian).

122. J. Ghosh. Multiclassifier systems: Back to the future. In F. Roli and J. Kittler, editors, *Proc. 3rd International Workshop on Multiple Classifier Systems, MCS'02*, Vol. 2364 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2002, Springer-Verlag, pp. 1–15.

123. E. M. Kleinberg. Stochastic discrimination. *Annals of Mathematics and Artificial Intelligence*, 1:207–239, 1990.

124. E. M. Kleinberg. On the algorithmic implementation of stochastic discrimination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(5):473–490, 2000.

125. L. Xu, A. Krzyzak, and C. Y. Suen. Methods of combining multiple classifiers and their application to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:418–435, 1992.

126. T. K. Ho, J. J. Hull, and S. N. Srihari. Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:66–75, 1994.

127. J. D. Tubbs and W. O. Alltop. Measures of confidence associated with combining classification rules. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:690–692, 1991.

128. W. H. E. Day. Consensus methods as tools for data analysis. In H. H. Bock, editor, *Classification and Related Methods for Data Analysis*, Elsevier Science Publishers B.V. (North Holland), 1988, pp. 317–324.

129. R. Battiti and A. M. Colla. Democracy in neural nets: Voting schemes for classification. *Neural Networks*, 7:691–707, 1994.

130. L. Lam and A. Krzyzak. A theoretical analysis of the application of majority voting to pattern recognition. In *12th International Conference on Pattern Recognition*, Jerusalem, Israel, 1994, pp. 418–420.

131. L. Lam and C. Y. Suen. Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(5):553–568, 1997.

132. X. Lin, S. Yacoub, J. Burns, and S. Simske. Performance analysis of pattern classifier combination by plurality voting. *Pattern Recognition Letters*, 24(12):1795–1969, 2003.

133. D. Ruta and B. Gabrys. A theoretical analysis of the limits of majority voting errors for multiple classifier systems. Technical Report 11, ISSN 1461-6122, Department of Computing and Information Systems, University of Paisley, December 2000.

134. L. Shapley and B. Grofman. Optimizing group judgemental accuracy in the presence of interdependencies. *Public Choice*, 43:329–343, 1984.

135. P. A. Lachenbruch. Multiple reading procedures: The performance of diagnostic tests. *Statistics in Medicine*, 7:549–557, 1988.

136. L. I. Kuncheva, C. J. Whitaker, C. A. Shipp, and R. P. W. Duin. Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis and Applications*, 6:22–31, 2003.

137. O. Matan. On voting ensembles of classifiers (extended abstract). In *Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models*, 1996, pp. 84–88.

138. M. Demirekler and H. Altincay. Plurality voting-based multiple classifier systems: statistically independent with respect to dependent classifiers sets. *Pattern Recognition*, 35:2365–2379, 2002.

139. W. Pierce. *Improving reliability of digital systems by redundancy and adaptation*. PhD thesis, Electrical Engineering, Stanford University, 1961.

140. D. M. Titterington, G. D. Murray, L. S. Murray, D. J. Spiegelhalter, A. M. Skene, J. D. F. Habbema, and G. J. Gelpke. Comparison of discriminant techniques applied to a complex data set of head injured patients. *Journal of the Royal Statistical Society, Series A* (*General*), 144(2):145–175, 1981.

141. P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero–one loss. *Machine Learning*, 29:103–130, 1997.

142. Y. S. Huang and C. Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:90–93, 1995.

143. K.-D. Wernecke. A coupling procedure for discrimination of mixed data. *Biometrics*, 48:497–506, 1992.

144. H.-J. Kang, K. Kim, and J. H. Kim. Approximating optimally discrete probability distribution with *k*th-order dependency for combining multiple decisions. *Information Processing Letters*, 62:67–75, 1997.

145. H.-J. Kang, K. Kim, and J. H. Kim. A framework for probabilistic combination of multiple classifiers at an abstract level. *Engineering Applications of Artificial Intelligence*, 10(4):379–385, 1997.

146. H. J. Kang, K. Kim, and J. H. Kim. Optimal approximation of discrete probability distribution with *k*th-order dependency and its application to combining multiple classifiers. *Pattern Recognition Letters*, 18:515–523, 1997.

147. C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):463–467, 1968.

148. C. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36:33–58, 1999.

149. L. I. Kuncheva, J. C. Bezdek, and R. P. W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34(2):299–314, 2001.

150. J. Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A. J. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, 2000, pp. 61–74.

151. W. Wei, T. K. Leen, and E. Barnard. A fast histogram-based postprocessor that improves posterior probability estimates. *Neural Computation*, 11(5):1235–1248, 1999.

152. B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *Proceedings of 18th International Conference on Machine Learning* (*ICML'01*), 2001, pp. 609–616.

153. J. Drish. *Obtaining Calibrated Probability Estimates from Support Vector Machines*, Technical Report, University of California, San Diego, 2001.

154. F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

155. K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

156. D. Dubois and H. Prade. A review of fuzzy set aggregation connectives. *Information Sciences*, 36:85–121, 1985.

157. R. P. W. Duin and D. M. J. Tax. Experiments with classifier combination rules. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000, Springer, pp. 16–29.

158. D. J. Miller and L. Yan. Critic-driven ensemble classification. *IEEE Transactions on Signal Processing*, 47(10):2833–2844, 1999.

159. D. M. J. Tax, R. P. W. Duin, and M. van Breukelen. Comparison between product and mean classifier combination rules. In *Proc. Workshop on Statistical Pattern Recognition*, Prague, Czech, 1997.

160. D. M. J. Tax, M. van Breukelen, R. P. W. Duin, and J. Kittler. Combining multiple classifier by averaging or multiplying? *Pattern Recognition*, 33:1475–1485, 2000.

161. R. R. Yager. On Ordered Weighted Averaging operators in multicriteria decision making. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, 18:183–193, 1988.

162. R. R. Yager and D. P. Filev. *Essentials of Fuzzy Modeling and Control*. John Wiley & Sons, N.Y, 1994.

163. L. I. Kuncheva. "Fuzzy" vs "non-fuzzy" in combining classifiers designed by boosting. *IEEE Transactions on Fuzzy Systems* 11:729–741, 2003.

164. A. Verikas, A. Lipnickas, K. Malmqvist, M. Bacauskiene, and A. Gelzinis. Soft combination of neural classifiers: A comparative study. *Pattern Recognition Letters*, 20:429–444, 1999.

165. G. Fumera and F. Roli. Performance analysis and comparison of linear combiners for classifier fusion. In *Proc. 16th International Conference on Pattern Recognition*, Canada, 2002.

166. S. Hashem. Optimal linear combinations of neural networks. *Neural Networks*, 10(4):599–614, 1997.

167. S. Hashem. Treating harmful collinearity in neural network ensembles. In A. J. C. Sharkey, editor, *Combining Artificial Neural Nets.* Springer-Verlag, London, 1999, pp. 101–125.

168. S. Hashem, B. Schmeiser, and Y. Yih. Optimal linear combinations of neural networks: an overview. In *IEEE International Conference on Neural Networks*, Orlando, Florida, 1994, pp. 1507–1512.

169. R. A. Jacobs. Methods for combining experts' probability assessments. *Neural Computation*, 7:867–888, 1995.

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

171. N. Ueda. Optimal linear combination of neural networks for improving classification performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(2):207–215, 2000.

172. S.-B. Cho. Pattern recognition with neural networks combined by genetic algorithm. *Fuzzy Sets and Systems*, 103:339–347, 1999.

173. L. Lam and C. Y. Suen. Optimal combination of pattern classifiers. *Pattern Recognition Letters*, 16:945–954, 1995.

174. J. A. Benediktsson, J. R. Sveinsson, J. I. Ingimundarson, H. Sigurdsson, and O. K. Ersoy. Multistage classifiers optimized by neural networks and genetic algorithms. *Nonlinear Analysis, Theory, Methods & Applications*, 30(3):1323–1334, 1997.

175. S.-B. Cho and J. H. Kim. Combining multiple neural networks by fuzzy integral and robust classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 25:380–384, 1995.

176. S. B. Cho and J. H. Kim. Multiple network fusion using fuzzy logic. *IEEE Transactions on Neural Networks*, 6:497–501, 1995.

177. P. D. Gader, M. A. Mohamed, and J. M. Keller. Fusion of handwritten word classifiers. *Pattern Recognition Letters*, 17:577–584, 1996.

178. A. Verikas and A. Lipnickas. Fusing neural networks through space partitioning and fuzzy integration. *Neural Processing Letters*, 16:53–65, 2002.

179. M. Grabisch. On equivalence classes of fuzzy connectives – the case of fuzzy integrals. *IEEE Transactions on Fuzzy Systems*, 3(1):96–109, 1995.

180. M. Grabisch, H. T. Nguen, and E. A. Walker. *Fundamentals of Uncertainty Calculi, With Applications to Fuzzy Inference*. Kluwer Academic Publishers, Dordrecht, 1995.

181. Y. S. Huang and C. Y. Suen. A method of combining multiple classifiers – a neural network approach. In *12th International Conference on Pattern Recognition*, Jerusalem, Israel, 1994, pp. 473–475.

182. L. I. Kuncheva. Using measures of similarity and inclusion for multiple classifier fusion by decision templates. *Fuzzy Sets and Systems*, 122(3):401–407, 2001.

183. C. Dietrich, F. Schwenker, and G. Palm. Classification of time series utilizing temporal and decision fusion. In J. Kittler and F. Roli, editors, *Proc. Second International Workshop on Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001, Springer-Verlag, pp. 378–387.

184. G. Giacinto, F. Roli, and L. Didaci. Fusion of multiple classifier for intrusion detection in computer networks. *Pattern Recognition Letters*, 24:1795–1803, 2003.

185. J. Kittler, M. Balette, J. Czyz, F. Roli, and L. Vanderdorpe. Decision level fusion of intramodal personal identity verification experts. In F. Roli and J. Kittler, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, Vol. 2364 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2002, Springer-Verlag, pp. 314–324.

186. Y. Lu. Knowledge integration in a multiple classifier system. *Applied Intelligence*, 6:75–86, 1996.

187. G. Rogova. Combining the results of several neural network classifiers. *Neural Networks*, 7:777–781, 1994.

188. R. F. Bordley. A multiplicative formula for aggregating probability assessments. *Management Science*, 28:1137–1148, 1982.

189. L. I. Kuncheva. On combining multiple classifiers. In *Proc. 7th International Conference on Information Processing and Management of Uncertainty* (*IPMU'98*), Paris, France, 1998, pp. 1890–1891.

190. T. Lee, J. A. Richards, and P. H. Swain. Probabilistic and evidential approaches for multisource data analysis. *IEEE* Transactions on Geoscience and Remote Sensing, 25(3):283–293, 1987.

191. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

192. P. Domingos. Why does bagging work? A Bayesian account and its implications. In Proc 3d International conference on *Knowledge Discovery and Data Mining*, Newport Beach, CA: AAAI Press, 1997, pp. 155–158.

193. A. Cedilnik and K. Košmelj. Relations among Fisher, Shannon–Wiener and Kullback measures of information for continuous variables. In A. Mrvar and A. Ferligoj, editors, *Developments in Statistics*. Metodološki zveski, Ljubljana: FDV, 2002.

194. C. Berenstein, L. N. Kanal, and D. Lavine. Consensus rules. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, Elsevier Science Publishers B.V., 1986, pp. 27–32.

195. J. A. Benediktsson and P. H. Swain. Consensus theoretic classification methods. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, 22:688–704, 1992.

196. K. -C. Ng and B. Abramson. Consensus diagnosis: A simulation study. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, 22:916–928, 1992.

197. E. Alpaydin and C. Kaynak. Cascading classifiers. *KYBERNETIKA*, 34(4):369–374, 1998.

198. L. Bruzzone and R. Cossu. A multiple-cascade-classifier system for a robust and partially unsupervised updating of land-cover maps. *IEEE Transactions on Geoscience and Remote Sensing*, 40(9):1984–1996, 2002.

199. M. Egmont-Petersen, W. R. M. Dassen, and J. H. C. Reiber. Sequential selection of discrete features for neural networks – A Bayesian approach to building a cascade. *Pattern Recognition Letters*, 20(11–13):1439–1448, 1999.

200. J. Gamma and P. Brazdil. Cascade generalization. *Machine Learning*, 41(3):315–343, 2000.

201. L. I. Kuncheva. Change-glasses approach in pattern recognition. *Pattern Recognition Letters*, 14:619–623, 1993.

202. G. Giacinto and F. Roli. Design of effective neural network ensembles for image classification processes. *Image Vision and Computing Journal*, 19(9–10):699–707, 2001.

203. L. I. Kuncheva. Clustering-and-selection model for classifier combination. In *Proc. Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, Brighton, UK, 2000, pp. 185–188.

204. L. I. Kuncheva. Switching between selection and fusion in combining classifiers: An experiment. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, 32(2):146–156, 2002.

205. R. Liu and B. Yuan. Multiple classifier combination by clustering and selection. *Information Fusion*, 2:163–168, 2001.

206. R. Avnimelech and N. Intrator. Boosted mixture of experts: an ensemble learning scheme. *Neural Computation*, 11:475–490, 1999.

207. A. Lazarevic and Z. Obradovic. Adaptive boosting techniques in heterogeneous and spatial databases. *Intelligent Data Analysis*, 5:1–24, 2001.

208. M. I. Jordan and R. A. Jacobs. Hierarchies of adaptive experts. In *International Conference on Neural Networks*, San Diego, California, 1992, pp. 192–196.

209. M. I. Jordan and L. Xu. Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, 8:1409–1431, 1995.

210. S. J. Nowlan and G. E. Hinton. Evaluation of adaptive mixtures of competing experts. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, 1991, pp. 774–780.

211. L. Breiman. Bagging predictors. Technical Report 421, Department of Statistics, University of California, Berkeley, 1994.

212. B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, N.Y, 1993.

213. 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*, 26(5):1651–1686, 1998.

214. L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

215. L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36:85–103, 1999.

216. N. V. Chawla, L. O. Hall, K. W. Bowyer, T. E. Moore Jr, and W. P. Kegelmeyer. Distributed pasting of small votes. In F. Roli and J. Kittler, editors, *Proc. 3rd International Workshop on Multiple Classifier Systems, MCS'02*, volume 2364 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2002, Springer-Verlag, pp. 51–62.

217. H. Drucker, C. Cortes, L. D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6:1289–1301, 1994.

218. M. Skurichina. *Stabilizing weak classifiers*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 2001.

219. Y. Freund and R. E. Schapire. A decision–theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

220. Y. Freund and R. E. Schapire. Discussion of the paper "Arcing Classifiers" by Leo Breiman. *Annals of Statistics*, 26(3):824–832, 1998.

221. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

222. E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

223. R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

224. M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 48:31–44, 2002.

225. J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–374, 2000.

226. R. E. Schapire. The boosting approach to machine learning. An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

227. L. Mason, P. L. Bartlet, and J. Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 38(3):243–255, 2000.

228. G. I. Webb. MultiBoosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

229. N. C. Oza. Boosting with averaged weight vectors. In T. Windeatt and F. Roli, editors, *Proc 4th Int. Workshop on Multiple Classifier Systems* (*MCS 2003*), Vol. 2709 of *Lecture Notes in Computer Science LNCS*, Guildford, UK, 2003. Springer Verlag, pp. 15–24.

230. P. M. Long and V. B. Vega. Boosting and microarray data. *Machine Learning*, 52:31–44, 2003.

231. R. Kohavi and D. H. Wolpert. Bias plus variance decomposition for zero–one loss functions. In L. Saitta, editor, *Machine Learning: Proc. 13th International Conference*, Morgan Kaufmann, 1996, pp. 275–283.

232. L. Breiman. Combining predictors. In A. J. C. Sharkey, editor, *Combining Artificial Neural Nets*, Springer-Verlag, London, 1999, pp. 31–50.

233. T. G. Dietterich. Bias-variance analysis of ensemble learning. In *Presented at the 7th Course of the International School on Neural Networks "E.R. Caianiello", Ensemble Methods for Learning Machines*, Vietri-sul-Mare, Salerno, Italy, 2002.

234. P. Domingos. A unified bias-variance decomposition and its applications. In *Proc. 7th International Conference on Machine Learning*, Stanford, CA, 2000, Morgan Kaufmann, pp. 231–238.

235. P. Domingos. A unified bias-variance decomposition for zero–one and squared loss. In *Proc. 7th International Conference on Artificial Intelligence*, Austin, TX, 2000, pp. 564–569.

236. E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–142, 1999.

237. T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization. *Machine Learning*, 40(2):139–157, 2000.

238. G. Valentini. *Ensemble methods based on bias-variance analysis*. PhD thesis, University of Genova, Genova, Italy, 2003.

239. J. R. Quinlan. Bagging, boosting and C4.5. In *Proc 13th Int. Conference on Artificial Intelligence AAAI-96*, Cambridge, MA, USA, 1996, pp. 725–730.

240. K. Chen, L. Wang, and H. Chi. Methods of combining multiple classifiers with different features and their applications to text-independent speaker identification. *International Journal on Pattern Recognition and Artificial Intelligence*, 11(3):417–445, 1997.

241. M. van Breukelen, R. P. W Duin, D. M. J. Tax, and J. E. den Hartog. Combining classifiers for the recognition of handwritten digits. In *1st IAPR TC1 Workshop on Statistical Techniques in Pattern Recognition*, Prague, Czech Republic, 1997, pp. 13–18.

242. R. Brill, R. Gutierrez-Osuna, and F. Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302, 2003.

243. T. K. Ho. The random space method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

244. E. Pękalska, M. Skurichina, and R. P. W. Duin. Combining Fisher linear discriminant for dissimilarity representations. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000, Springer, pp. 230–239.

245. L. I. Kuncheva. Genetic algorithm for feature selection for parallel classifiers. *Information Processing Letters*, 46:163–168, 1993.

246. L. I. Kuncheva and L. C. Jain. Designing classifier fusion systems by genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(4):327–336, 2000.

247. N. Oza and K. Tumer. Input decimation ensembles: Decorrelation through dimensionality reduction. In *Proc. 12nd International Workshop on Multiple Classifier Systems, MCS'01*, In Vol. 2096, *Lecture Notes in Computer Science*, Cambridge, UK, 2001, pp. 238–247.

248. D. W. Aha and R. L. Bankert. A comparative evaluation of sequential feature selection algorithms. In *Proc. 5th International Workshop on AI and Statistics*, Ft Lauderdale, FL, 1995, pp. 1–7.

249. P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125, 1994.

250. S. Puuronen, A. Tsymbal, and I. Skrypnyk. Correlation-based and contextual merit-based ensemble feature selection. In *Proc. 4th International Conference on Advances in Intelligent Data Analysis, IDA'01*, Vol. 2189 of *Lecture Notes in Computer Science*, Cascais, Portugal, 2001, pp. 135–144.

251. A. Tsymbal, S. Puuronen, and D. W. Patterson. Ensemble feature selection with the simple Bayesian classification. *Information Fusion*, 4:87–100, 2003.

252. S. Günter and H. Bunke. Creation of classifier ensembles for handwritten word recognition using feature selection algorithms. In *Proc. 8th International Workshop on Frontiers in Handwriting Recognition*, Canada, 2002.

253. D. W. Opitz. Feature selection for ensembles. In *Proc. 16th National Conference on Artificial Intelligence*, (*AAAI*), Orlando, FL, 1999, pp. 379–384.

254. D. W. Aha and R. L. Blankert. Cloud classification using error-correcting output codes. *Artificial Intelligence Applications: Natural Resources*, *Agriculture and Environmental Science*, 11(1):13–28, 1997.

255. T. G. Dietterich and G. Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proc 9th National Conference on Artificial Intelligence, AAAI-91*, AAAI Press, 1991, pp. 572–577.

256. T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

257. F. Masulli and G. Valentini. Comparing decomposition methods for classification. In *Proc. International Conference on Knowledge-Based Intelligent Engineering Systems and Applied Technologies* (*Kes 2000*), Brighton, UK, 2000, pp. 788–792.

258. F. Masulli and G. Valentini. Effectiveness of error-correcting output codes in multiclass learning problems. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000. Springer, pp. 107–116.

259. R. E. Shapire. Using output codes to boost multiclass learning problems. In *Proc. 14th International Conference on Machine Learning*, 1997.

260. F. Masulli and G. Valentini. Dependence among codeword bits errors in ECOC learning machines: An experimental analysis. In *Proc. 2nd International Workshop on Multiple*

Classifier Systems (*MCS 2001*), Vol. 2096 in *Lecture Notes in Computer Science*, Cambridge, UK, 2001, pp. 158–167.

261. E. B. Kong and T. G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proc. 12th International Conference on Machine Learning*, Morgan Kaufmann, CA, USA, 1995, pp. 313–321.
262. F. Cutzu. Polychotomous classification with pairwise classifiers: A new voting principle. In *Proc. 4th International Workshop on Multiple Classifier Systems (MCS 2003), Lecture Notes in Computer Science*, Guildford, UK, 2003, Vol. 2709, pp. 115–124.
263. A. Jóźwik and G. Vernazza. Recognition of leucocytes by a parallel k-nn classifier. In *Proc. International Conference on Computer-Aided Medical Diagnosis*, Warsaw, 1987, pp. 138–153.
264. M. Pardo, G. Sberveglieri, F. Masulli, and G. Valentini. Decompositive classification models for electronic noses. *Analitica Chimica Acta*, 446:223–232, 2001.
265. A. L. N. Fred and A. K. Jain. Robust data clustering. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, USA, 2003.
266. A. Strehl and J. Ghosh. Cluster ensembles – A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–618, 2002.
267. A. Strehl and J. Ghosh. Cluster ensembles – A knowledge reuse framework for combining partitionings. In *Proceedings of AAAI*. AAAI/MIT Press, 2002.
268. W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.
269. A. Ben-Hur, A. Elisseeff, and I. Guyon. A stability based method for discovering structure in clustered data. In *Proc. Pacific Symposium on Biocomputing*, 2002, pp. 6–17.
270. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.
271. H. Ayad and M. Kamel. Finding natural clusters using multi-clusterer combiner based on shared nearest neighbors. In T. Windeatt and F. Roli, editors, *Proc. 4th International Workshop on Multiple Classifier Systems, MCS'03*, volume 2709 of *Lecture Notes in Computer Science*, Guildford, UK, 2003, Springer-Verlag, pp. 166–175.
272. A. Fred. Finding consistent clusters in data partitions. In F. Roli and J. Kittler, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, *MCS'01*, Vol. 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001, Springer-Verlag, pp. 309–318.
273. A. Fred and A. K. Jain. Data clustering using evidence accumulation. In *Proc. 16th International Conference on Pattern Recognition, ICPR*, Canada, 2002, pp. 276–280.
274. V. Di Gesu. Integrated fuzzy clustering. *Fuzzy Sets and Systems*, 68:293–308, 1994.
275. S. Monti, P. Tamayo, J. Mesirov, and T. Golub. Consensus clustering: A resampling based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52:91–118, 2003.
276. A. Weingessel, E. Dimitriadou, and K. Hornik. An ensemble method for clustering, 2003. Working paper, http://www.ci.tuwien.ac.at/Conferences/DSC-2003/.
277. W. L. Winston. *Operations Research. Algorithms and Applications*, 3rd Ed. International Thomson Publishing, USA, 1994.
278. K. Tumer and J. Ghosh. Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348, 1996.

279. K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3/4):385–404, 1996.

280. K. Tumer and J. Ghosh. Linear and order statistics combiners for pattern classification. In A. J. C. Sharkey, editor, *Combining Artificial Neural Nets.* Springer-Verlag, London, 1999, pp. 127–161.

281. G. Fumera and F. Roli. Linear combiners for classifier fusion: some theoretical and experimental results. In T. Windeatt and F. Roli, editors, *Proc. 4th Int. Workshop on Multiple Classifier Systems* (*MCS 2003*), Vol. 2709 of *Lecture Notes in Computer Science LNCS*, Guildford, UK, 2003, Springer-Verlag, pp. 74–83.

282. F. M. Alkoot and J. Kittler. Experimental evaluation of expert fusion strategies. *Pattern Recognition Letters*, 20:1361–1369, 1999.

283. L. I. Kuncheva. A theoretical study on expert fusion strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):281–286, 2002.

284. A. M. Mood, F. A. Graybill, and D. C. Boes. *Introduction to the Theory of Statistics*, 3rd Ed. McGraw-Hill Series in Probability and Statistics. McGraw-Hill Inc., 1974.

285. J. Kittler and F. M. Alkoot. Relationship of sum and vote fusion strategies. In F. Roli and J. Kittler, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems, MCS'01*, Vol. 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001, Springer-Verlag, pp. 339–348.

286. D. Chen and X. Cheng. An asymptotic analysis of some expert fusion methods. *Pattern Recognition Letters*, 22:901–904, 2001.

287. C. R. Rao. Diversity: Its measurement, decomposition, apportionment and analysis. *Sankya: The Indian Journal of Statistics, Series A*, 44(1):1–22, 1982.

288. R. E. Hampton. *Biological Statistics*. Wm. C. Brown Communications, USA, 1994.

289. D. E. Eckhardt and L. D. Lee. A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Transactions on Software Engineering*, 11(12):1511–1517, 1985.

290. B. Littlewood and D. R. Miller. Conceptual modeling of coincident failures in multiversion software. *IEEE Transactions on Software Engineering*, 15(12):1596–1614, 1989.

291. D. Partridge and W. J. Krzanowski. Software diversity: practical statistics for its measurement and exploitation. *Information & Software Technology*, 39:707–717, 1997.

292. D. Partridge and W. Krzanowski. Refining multiple classifier system diversity. Technical Report 348, Computer Science Department, University of Exeter, UK, 2003.

293. P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. W. H. Freeman & Co, 1973.

294. G. U. Yule. On the association of attributes in statistics. *Phil. Trans., A*, 194:257–319, 1900.

295. D. B. Skalak. The sources of increased accuracy for two proposed boosting algorithms. In *Proc. American Association for Artificial Intelligence*, *AAAI-96*, *Integrating Multiple Learned Models Workshop*, 1996.

296. D. Ruta and B. Gabrys. Analysis of the correlation between majority voting error and the diversity measures in multiple classifier systems. In *Proc. SOCO 2001*, Paisley, Scotland, 2001.

297. P. Cunningham and J. Carney. Diversity versus quality in classification ensembles based on feature selection. Technical Report TCD-CS-2000-02, Department of Computer Science, Trinity College, Dublin, 2000.

298. L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

299. M. Aksela. Comparison of classifier selection methods for improving committee performance. In T. Windeatt and F. Roli, editors, *Proc. 4th Int. Workshop on Multiple Classifier Systems* (*MCS 2003*), *Lecture Notes in Computer Science LNCS* **2709**, Guildford, UK, 2003, Springer Verlag, pp. 84–93.

300. R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. A new ensemble diversity measure applied to thinning ensembles. In T. Windeatt and F. Roli, editors, *Proc. 4th Int. Workshop on Multiple Classifier Systems* (*MCS 2003*), *Lecture Notes in Computer Science LNCS* **2709**, Guildford, UK, 2003, Springer Verlag, pp. 306–316.

301. D. Ruta. *Classifier diversity in combined pattern recognition systems*. PhD thesis, University of Paisley, Scotland, UK, 2003.

302. L. I. Kuncheva and R. K. Kountchev. Generating classifier outputs of fixed accuracy and diversity. *Pattern Recognition Letters*, 23:593–600, 2002.

303. L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles. *Machine Learning*, 51:181–207, 2003.

304. E. Pękalska, R.P.W. Duin, and M. Skurichina. A discussion on the classifier projection space for classifier combining. In F. Roli and J. Kittler, editors, *Proc. 3rd International Workshop on Multiple Classifier Systems*, *MCS'02*, Vol. 2364 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2002, Springer-Verlag, pp. 137–148.

305. D. D. Margineantu and T. G. Dietterich. Pruning adaptive boosting. In *Proc. 14th International Conference on Machine Learning*, San Francisco, 1997, Morgan Kaufmann, pp. 378–387.

306. G. Giacinto and F. Roli. An approach to the automatic design of multiple classifier systems. *Pattern Recognition Letters*, 22:25–33, 2001.

307. F. Roli, G. Giacinto, and G. Vernazza. Methods for designing multiple classifier systems. In J. Kittler and F. Roli, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, Vol. 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001, Springer-Verlag, pp. 78–87.

308. A. J. C. Sharkey, N. E. Sharkey, U. Gerecke, and G. O. Chandroth. The test-and-select approach to ensemble combination. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, Vol. 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000, Springer, pp. 30–44.

309. D. Ruta and B. Gabrys. Application of the evolutionary algorithms for classifier selection in multiple classifier systems with majority voting. In J. Kittler and F. Roli, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, Vol. 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001, Springer-Verlag, pp. 399–408.

310. K. Sirlantzis, M. C. Fairhurst, and M. S. Hoque. Genetic algorithms for multi-classifier system configuration: a case study in character recognition. In J. Kittler and F. Roli, editors, *Proc. 2nd International Workshop on Multiple Classifier Systems*, Vol. 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001, Springer-Verlag, pp. 99–108.

311. C. A. Shipp and L. I. Kuncheva. Relationships between combination methods and measures of diversity in combining classifiers. *Information Fusion*, 3(2):135–148, 2002.

312. R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

Index

Combining Pattern Classifiers: Methods and Algorithms, by Ludmila I. Kuncheva
ISBN 0-471-21078-1