

Image Processing, Analysis, and Machine Vision

Fourth Edition

Milan Sonka

The University of Iowa, Iowa City

Vaclav Hlavac

Czech Technical University, Prague

Roger Boyle

Prifysgol Aberystwyth, Aberystwyth



Australia • Brazil • Japan • Mexico • Singapore • United Kingdom • United States



**Image Processing, Analysis, and
Machine Vision, Fourth Edition**
Milan Sonka, Vaclav Hlavac, and
Roger Boyle

Publisher, Global Engineering:
Timothy L. Anderson

Senior Developmental Editor:
Hilda Gowans

Senior Editorial Assistant: Tanya Altieri

Art and Cover Direction, Production
Management: PreMediaGlobal

Compositor: Hrvoje Bugonovic

Rights Acquisition Director:
Audrey Pettengill

Rights Acquisition Specialist,
Text and Image: Amber Hosea

Text & Image Permissions Researcher:
Kristiina Paul

Manufacturing Planner: Doug Wilke

Cover Image: © Milan Sonka,
Vaclav Hlavac, and Roger Boyle

© 2015, 2008 Cengage Learning

WCN: 02-200-203

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the publisher.

For product information and technology assistance, contact us at
Cengage Learning Customer & Sales Support, 1-800-354-9706

For permission to use material from this text or product,
submit all requests online at **www.cengage.com/permissions**
Further permissions questions can be emailed to
permissionrequest@cengage.com

Library of Congress Control Number: 2013947420

ISBN-13: 978-1-133-59360-7

ISBN-10: 1-133-59360-7

Cengage Learning

200 First Stamford Place, 4th Floor
Stamford, CT 06902
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at **www.cengage.com/global**

Cengage Learning products are represented in Canada by
Nelson Education, Ltd.

To learn more about Cengage Learning, visit
www.cengage.com/engineering.

Purchase any of our products at your local college store or at our
preferred online store **www.cengagebrain.com**

Printed in the United States of America
1 2 3 4 5 6 7 17 16 15 14 13

Chapter 9

Object recognition

Even the simplest machine vision tasks require the help of recognition. Basic methods of pattern recognition must be understood in order to study more complex machine vision processes leading to region and object classification. Classification of objects or regions has been mentioned several times; recognition is then the last step of the bottom-up image processing approach, and is often also used in other control strategies for image understanding.

Consider a simple recognition problem. Two different parties take place at the same hotel at the same time—the first is a celebration of a successful basketball season, and the second a yearly meeting of jockeys. The receptionist is giving directions to guests, asking which party they are to attend, but quickly learns that no questions are necessary and directs the guests to the right places, noticing that instead of questions, the obvious physical features of basketball players and jockeys can be used. Maybe just two features are used to make a decision: weight and height. All small and light men are directed to the jockey party, all tall and heavier guests are sent to the basketball party. So the early guests answered the doorman's question as to which party they are going to visit and this information, together with characteristic features of these guests, resulted in the ability to classify them based only on their features. Plotting height and weight in a two-dimensional space (see Figure 9.1), it is clear that jockeys and basketball players form two easily separable classes and that this recognition task is extremely simple. Real object recognition problems are often more difficult, and the classes do not differ so substantially, but the principles remain the same.

Pattern recognition is discussed at length in many references [Duda and Hart, 1973; Haykin, 1998; Duda et al., 2000; Bishop, 2006], and only a brief introduction will be given here. We will also introduce some other related techniques: graph matching, neural nets, genetic algorithms, simulated annealing, and fuzzy logic. Many open-source pattern recognition packages have become available; the Weka project (in Java) and pattern recognition for the **R** statistical package are perhaps the most prominent [Weka 3 Team, 2013; R Development Core Team, 2008; R Software Team, 2013]. Since many pattern recognition tasks can be efficiently solved by employing such well-developed and well-tested tools, becoming proficient with their use may be of interest.

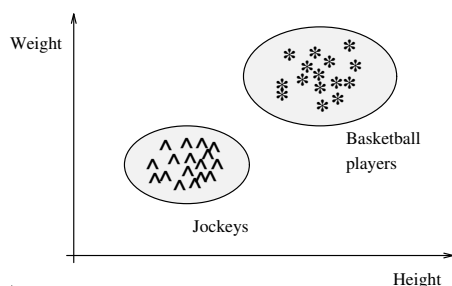


Figure 9.1: Recognition of basketball players and jockeys. © Cengage Learning 2015.

No recognition is possible without knowledge. Decisions about classes into which objects are classified are based on such knowledge. Both specific knowledge about the objects being processed and hierarchically higher and more general knowledge about object classes is required.

9.1 Knowledge representation

Here we present a short outline of common knowledge representation techniques and representations as they are used in AI, and an overview of some basic knowledge representations. More detailed coverage can be found in [Michalski et al., 1983; Wechsler, 1990; Reichelt, 1991; Sowa, 1999; Brachman and Levesque, 2004].

A good knowledge representation design is the most important part of solving the understanding problem and a small number of relatively simple control strategies is often sufficient for AI systems to show complex behavior, assuming an appropriately complex knowledge base is available. In other words, given a rich, well-structured representation of a large set of a priori data and hypotheses a high degree of control sophistication is not required for intelligent behavior.

Other terms of which regular use will be made are **syntax** and **semantics** [Winston, 1984]. The **syntax** of a representation specifies the symbols that may be used and the ways that they may be arranged while the **semantics** of a representation specifies how meaning is embodied in the symbols and the symbol arrangement allowed by the syntax. A **representation** is then a set of syntactic and semantic conventions that make it possible to describe things.

The main knowledge representation techniques used in AI are formal grammars and languages, predicate logic, production rules, semantic nets, and frames. Note that knowledge representation data structures are mostly extensions of conventional data structures such as lists, trees, graphs, tables, hierarchies, sets, rings, nets, and matrices.

Descriptions, features

Descriptions and features are not pure knowledge representations but can nevertheless be used for representing knowledge as a part of a more complex structure. Descriptions usually represent some scalar properties of objects, and are called **features**. Typically, several descriptions are combined into **feature vectors**. Numerical feature vectors are inputs for statistical pattern recognition techniques (see Section 9.2).

Grammars, languages

Feature description is not adequate to describe an object's structure; a structural description is formed from existing primitives (elementary structural properties of the objects) and relations between them.

The simplest form of structure representations are chains, trees, and general graphs. A well-known example is structural description of chromosomes using border segments as primitives [Fu, 1982] (see Figure 8.15), where borders are represented by a chain of symbols, the symbols representing specific types of border primitives. Hierarchical structures can be represented by trees—the concavity tree of Figure 8.31 is an example. A more general graph representation is used in Chapter 15, where a graph grammar (Figure 15.11) is used for texture description. Many examples of syntactic object description may be found in [Fu, 1982].

One object can be described by a chain, a tree, a graph, etc., of symbols. Nevertheless, the whole class of objects cannot be described by a single chain, a single tree, etc., but a class of structurally described objects can be represented by **grammars** and **languages**. Grammars and languages provide rules defining how the chains, trees, or graphs can be constructed from a set of primitives. A more specific description of grammars and languages is given in Section 9.4.

Predicate logic

Predicate logic plays a very important role in knowledge representation—it introduces a mathematical formalism to derive new knowledge from old knowledge by applying mathematical deduction. Predicate logic works with combinations of logic variables, quantifiers ($\exists$, $\forall$), and logic operators (*and*, *or*, *not*, *implies*, *equivalent*). The logic variables are binary (*true*, *false*). The idea of proof and rules of inference such as **modus ponens** and **resolution** are the principal building blocks.

Predicate logic forms the essence of the programming language PROLOG, which is widely used if objects are described by logic variables. Requirements of 'pure truth' represent the main weakness of predicate logic in knowledge representation, since it does not allow uncertain or incomplete information.

Production rules

Production rules represent a wide variety of knowledge representations that are based on **condition action** pairs. The essential model of behavior of a system based on production rules (a production system) can be described as follows:

if condition *X* holds *then* action *Y* is appropriate.

Information about what action is appropriate at what time represents knowledge. The procedural character of knowledge represented by production rules is another important property—not all the information about objects must be listed as an object property. Consider a simple knowledge base where the following knowledge is presented using a production rule:

if ball *then* circular. (9.1)

and suppose the knowledge base also includes the statements

$$\begin{aligned} \text{object A } & \textit{is_a} \text{ ball,} \\ \text{object B } & \textit{is_a} \text{ ball,} \\ \text{object C } & \textit{is_a} \text{ shoe,} \\ & \text{etc.} \end{aligned} \tag{9.2}$$

To answer the question *how many objects are circular?*, if enumerative knowledge representation is used, the knowledge must be listed as

$$\begin{aligned} \text{object A } & \textit{is_a} \text{ (ball, circular),} \\ \text{object B } & \textit{is_a} \text{ (ball, circular),} \\ & \text{etc.} \end{aligned} \tag{9.3}$$

If procedural knowledge is used, the knowledge base (equation 9.2) together with the knowledge (equation 9.1) gives the same information in a significantly more efficient manner.

Both production rule knowledge representation and production systems appear frequently in computer vision and image understanding problems. Furthermore, production systems, together with a mechanism for handling uncertainty information, form a basis of expert systems.

Fuzzy logic

Fuzzy logic was developed [Zadeh, 1965; Zimmermann et al., 1984] to overcome the obvious limitations of numerical or crisp representation of information. Consider the use of knowledge represented by production rule (equation 9.1) for recognition of balls; using the production rule, the knowledge about balls may be represented as

$$\textit{if circular then ball.}$$

If the object in a two-dimensional image is considered circular then it may represent a ball. Our experience with balls, however, says that they are usually close to, but not perfectly, circular. Thus, it is necessary to define some circularity threshold so that all *reasonably* circular objects from our set of objects are labeled as balls. Here is the fundamental problem of crisp descriptions: how circular must an object be to be considered circular?

If humans represent such knowledge, the rule for ball circularity may look like

$$\textit{if circularity is HIGH then object is a ball with HIGH confidence.}$$

Clearly, high circularity is a preferred property of balls. Such knowledge representation is very close to common sense representation of knowledge, with no need for exact specification of the circularity/non-circularity threshold. **Fuzzy rules** are of the form

$$\textit{if X is A then Y is B,}$$

where X and Y represent some properties and A and B are **linguistic variables**. Fuzzy logic can be used to solve object recognition and other decision-making tasks, among others; this is discussed further in Section 9.7.

Semantic nets

Semantic nets are a special variation of relational data structures (see Chapter 4). The semantics distinguish them from general nets—semantic nets consist of objects, their description, and a description of relations between objects (often just relations between neighbors). Logical forms of knowledge can be included in semantic nets, and predicate logic can be used to represent and/or evaluate the local information and local knowledge. Semantic nets can also represent common sense knowledge that is often imprecise and needs to be treated in a probabilistic way. Semantic nets have a hierarchical structure; complex representations consist of less complex representations, which can in turn be divided into simpler ones, etc. Relations between partial representations are described at all appropriate hierarchical levels.

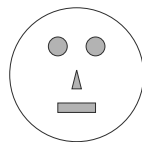


Figure 9.2: A human face: A face is a circular part of the human body that consists of two eyes, one nose, and one mouth; One eye is positioned left of the other eye; The nose is between and below the eyes; The mouth is below the nose; An eye is approximately circular; The nose is vertically elongated; The mouth is horizontally elongated. © Cengage Learning 2015.

Semantic nets are represented by labeled graphs: nodes represent objects and arcs represent relations between them. A simple example is illustrated in Figure 9.2, and its associated semantic net is shown in Figure 9.3.

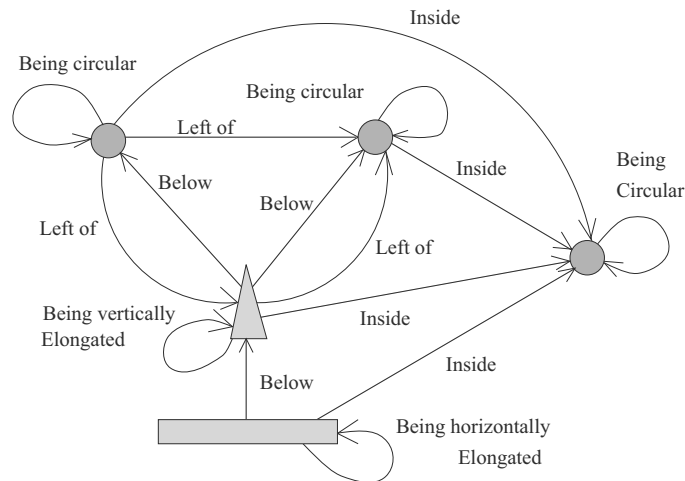


Figure 9.3: Semantic net of a human face. © Cengage Learning 2015.

The descriptive structures found in real images match the knowledge represented by a semantic net with varying degrees of closeness. How well such structures are represented by a semantic net is discussed in Section 9.5 and Chapter 10.

A detailed discussion of semantic nets related to image information can be found in [Niemann, 1990], and more general properties of semantic nets are described in [Michalski et al., 1983; Sharples et al., 1989].

Frames, scripts

Frames—which are considered high-level knowledge representation—provide a very general method for knowledge representation which may contain all the knowledge representation principles discussed so and is suitable for representing common sense knowledge under specific circumstances. They are sometimes called **scripts** because of their similarity to film scripts. Consider a frame called *plane_start*, which may consist of the following sequence of actions:

1. Start the engines.
2. Taxi to the runway.
3. Increase RPMs of engines to maximum.
4. Travel along runway increasing speed.
5. Fly.

Assuming this frame represents knowledge of how planes *usually* start, the situation of a plane standing on a runway with engines running causes the prediction that the plane will start in a short time. The frame can be used as a substitute for missing information which may be extremely important in vision-related problems.

Assuming that one part of the runway is not visible from the observation point, using the *plane_start* frame, a computer vision system can overcome the lack of continuous information between the plane moving at the beginning of the runway and flying when it next appears. If it is a passenger plane, the frame may have additional items such as *time of departure*, *time of arrival*, *departure city*, *arrival city*, *airline*, *flight number*, etc., because in a majority of cases it makes sense to be interested in this information if we identify a passenger plane.

From a formal point of view, a frame is represented by a general semantic net accompanied by a list of relevant variables, concepts, and concatenation of situations. No standard form of frame exists. Frames represent a tool for organizing knowledge in prototypical objects, and for description of mutual influences of objects using stereotypes of behavior in specific situations. Examples of frames can be found elsewhere [Michalski et al., 1983; Sharples et al., 1989].

9.2 Statistical pattern recognition

A physical object is usually represented in image analysis and computer vision by a region in a segmented image. The set of objects can be divided into disjoint subsets, that, from the classification point of view, have some common features and are called **classes**. How objects are divided into classes is imprecise and depends on the classification goal.

Object recognition assigns classes to objects, and the algorithm that does this is called a **classifier**. The number of classes is usually known beforehand, and typically can be derived from the problem specification. Nevertheless, there are approaches in which the number of classes may not be known (see Section 9.2.6).

The classifier (similarly to a human) does not decide about the class from the object itself—rather, sensed object properties serve this purpose. For example, to distinguish steel from sandstone, we do not have to determine molecular structures, even though

this would describe them well. Properties such as texture, specific weight, hardness, etc., are used instead. These properties are called the **pattern**, and the classifier actually recognizes the patterns and not the objects. Object recognition and pattern recognition can be considered synonymous.

The main pattern recognition steps are shown in Figure 9.4. The block ‘Construction of formal description’ is based on the experience and intuition of the designer. A set of elementary properties is chosen which describe some characteristics of the object; these properties are measured in an appropriate way and form the description pattern of the object. These properties can be either quantitative or qualitative in character and their form can vary (numerical vectors, chains, etc.). The theory of recognition deals with the problem of designing the classifier for the specific (chosen) set of elementary object descriptions.

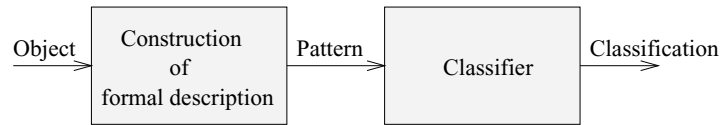


Figure 9.4: Main pattern recognition steps. © Cengage Learning 2015.

Statistical object description uses elementary numerical descriptions called **features**, $x_1, x_2, \dots, x_n$; in image analysis, the features result from object description as discussed in Chapter 8. The pattern (also referred to as pattern vector, or feature vector) $\mathbf{x} = (x_1, x_2, \dots, x_n)$ that describes an object is a vector of elementary descriptions, and the set of all possible patterns forms the **pattern space** X (also called **feature space**). If the elementary descriptions are chosen well, patterns of objects of each class are ‘close’ in pattern space. The classes form clusters in the feature space, which can be separated by a discrimination curve (or hyper-surface in a multi-dimensional feature space)—see Figure 9.5.

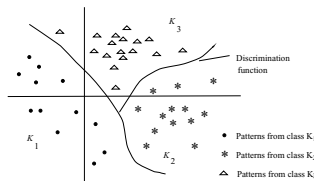


Figure 9.5: Discrimination functions between three classes. © Cengage Learning 2015.

If a discrimination hyper-surface exists which separates the feature space such that only objects from one class are in each separated region, the problem is called a recognition task with **separable classes**. If the discrimination hyper-surfaces are hyper-planes, it is called a **linearly separable** task. Intuitively, we may expect that separable classes can be recognized without errors.

The majority of object recognition problems do not have separable classes, in which case the locations of the discrimination hyper-surfaces in the feature space can never separate the classes correctly and some objects will always be misclassified.

9.2.1 Classification principles

A statistical classifier is a device with n inputs representing the n features $x_1, x_2, \dots, x_n$ measured from an object to be classified, and 1 output. An R -class classifier will generate one of R symbols $\omega_1, \omega_2, \dots, \omega_R$ as an output, and the user interprets this output as a decision about the class of the processed object. The generated symbols ω_r are the **class identifiers**.

A **decision rule** is a function $d(\mathbf{x}) = \omega_r$ which describes relations between classifier inputs and the output. The decision rule divides the feature space into R disjoint subsets K_r , $r = 1, \dots, R$, each of which includes all the feature representation vectors $\mathbf{x}'$ of objects for which $d(\mathbf{x}') = \omega_r$. The borders between subsets K_r , $r = 1, \dots, R$ form the discrimination hyper-surfaces. The determination of discrimination hyper-surfaces (or definition of the decision rule) is the goal of classifier design.

The discrimination hyper-surfaces can be defined by R scalar functions $g_1(\mathbf{x})$, $g_2(\mathbf{x})$, $\dots$, $g_R(\mathbf{x})$ called **discrimination functions**. $g_r(\mathbf{x})$ is maximal for $\mathbf{x} \in K_r$:

$$g_r(\mathbf{x}) \geq g_s(\mathbf{x}), s \neq r. \quad (9.4)$$

Then the discrimination hyper-surface between class regions K_r and K_s is defined by

$$g_r(\mathbf{x}) - g_s(\mathbf{x}) = 0. \quad (9.5)$$

The decision rule results from this definition:

$$d(\mathbf{x}) = \omega_r \iff g_r(\mathbf{x}) = \max_{s=1, \dots, R} g_s(\mathbf{x}). \quad (9.6)$$

Linear discrimination functions are the simplest and are widely used. Their general form is

$$g_r(\mathbf{x}) = q_{r0} + q_{r1}x_1 + \dots + q_{rn}x_n \quad (9.7)$$

for all $r = 1, \dots, R$. If all the discrimination functions of the classifier are linear, it is called a **linear classifier**.

Non-linear classifiers usually transform the original feature space X^n into a new feature space X^m applying some appropriate non-linear function Φ , where the superscripts n, m refer to the space dimensionality

$$\Phi = (\phi_1, \phi_2, \dots, \phi_m) : X^n \rightarrow X^m. \quad (9.8)$$

After the transformation, a linear classifier is applied in the new feature space—the role of the function Φ is to ‘straighten’ the non-linear discrimination hyper-surfaces of the original space into hyper-planes in the transformed space. The discrimination functions of such a classifier are

$$g_r(\mathbf{x}) = q_{r0} + q_{r1}\phi_1(\mathbf{x}) + \dots + q_{rm}\phi_m(\mathbf{x}), \quad (9.9)$$

where $r = 1, \dots, R$. We may re-write the formula in vector representation

$$g_r(\mathbf{x}) = \mathbf{q}_r \cdot \Phi(\mathbf{x}), \quad (9.10)$$

where $\mathbf{q}_r$, $\Phi(\mathbf{x})$ are vectors consisting of $q_{r0}, \dots, q_{rm}$ and $\phi_0(\mathbf{x}), \dots, \phi_m(\mathbf{x})$, respectively, $\phi_0(\mathbf{x}) \equiv 1$. Non-linear classifiers are described in detail in [Sklansky, 1981; Devijver and Kittler, 1982]. This important idea is revisited in Section 9.2.5 in considering support vector machines.

9.2.2 Nearest neighbors

We may construct classifiers based on the **minimum distance** principle; the resulting classifier is just a special case of classifiers with discrimination functions, but has computational advantages and may easily be implemented on digital computers. Assume that R points are defined in the feature space, $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_R$ that represent **exemplars** (sample patterns) of classes $\omega_1, \omega_2, \dots, \omega_R$. A minimum distance classifier classifies a pattern $\mathbf{x}$ into the class to whose exemplar it is closest.

$$d(\mathbf{x}) = \omega_r \iff |\mathbf{v}_r - \mathbf{x}| = \min_{s=1, \dots, R} |\mathbf{v}_s - \mathbf{x}|. \quad (9.11)$$

Discrimination hyper-planes are perpendicular to and bisect line segments $\mathbf{v}_s \mathbf{v}_r$ (Figure 9.6). If each class is represented by just one exemplar, a linear classifier results; one implementation of this is:

Algorithm 9.1: Minimum distance classifier learning and classification

1. Learning: For all classes, compute class exemplars $\mathbf{v}_i$ based on the training set

$$\mathbf{v}_i(k_i + 1) = \frac{1}{k_i + 1} \left(k_i \mathbf{v}_i(k_i) + \mathbf{x}_i(k_i + 1) \right),$$

where $\mathbf{x}_i(k_i + 1)$ are objects from the class i and k_i denotes the number of objects from class i used thus far for learning.

2. Classification: For an object description vector $\mathbf{x}$, determine the distance of $\mathbf{x}$ from the class exemplars $\mathbf{v}_i$. Classify the object into the class j if the distance of $\mathbf{x}$ from $\mathbf{v}_j$ is the minimum such (equation 9.11).

If more than one exemplar represents some class, the classifier gives piecewise linear discrimination hyper-planes—this is the elementary but widely applied **nearest neighbor** or NN classifier, which classifies an unseen pattern into the class of the nearest known training pattern. This simple idea is often made more resilient by allocating the new pattern to the most populous class of the k closest training patterns—this a k -NN classifier.

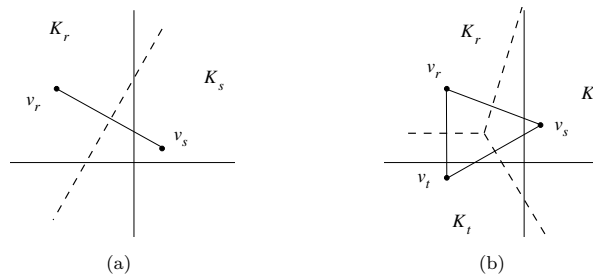


Figure 9.6: Minimum distance discrimination functions. © Cengage Learning 2015.

The nearest neighbor principle is very widely used in computer vision; often it is computed in some feature space as we outline here, and sometimes in connection with 2D or 3D spatial issues. It will be clear that for large data sets, especially if the dimensionality of the feature space is high—as it often is—the computation of nearest neighbor information may be very expensive. If the data come, for example, from a video stream then the expensive computation may need repeating with high frequency. The fundamental problem is that distance computation is costly and the total cost is unavoidably linear in the number of points compared. To increase the processing speed, it is possible to partition the data space and reduce this number significantly using K-D trees [Friedman et al., 1977]. If the data are in N dimensions, this data structure recursively partitions it on the co-ordinate which exhibits maximum variation: the resulting tree has leaves which are ‘small’ in densely occupied areas, and large in more sparse ones. The NN search is performed firstly in the leaf occupied by a test vector, and then only in neighboring leaves which contain plausible candidates.

Algorithm 9.2: Nearest neighbor search with K-D trees

1. For an N -dimensional dataset $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ determine the dimension i , $1 \leq i \leq N$ which exhibits greatest variation. Cut the data with a plane at its median value along that dimension.
2. Recurse this procedure to create a balanced binary tree of depth d .
3. To locate the NN of an input vector $\mathbf{y}$, determine which leaf cell it lies within. Perform an exhaustive search within this cell.
4. Traverse neighboring cells that could contain an element closer to $\mathbf{y}$ than the current best candidate.

Since the exhaustive search is performed at a small number of leaves (perhaps only one), this algorithm will run in $\log(n)$ time and is a great improvement.

Algorithm 9.2 works well for low N , but the number of neighbors to search can become high as N increases. For these reasons, a selection of *approximate* nearest neighbor algorithms is available that most of the time generate solutions that, if imperfect, are near optimal: at simplest, we may terminate after step 3. Very often this delivers the NN but if not, it is likely to be close.

Better approximate algorithms rely on selectively applying step 4—Beis and Lowe [Beis and Lowe, 1997] have developed a Best Bin First (BBF) algorithm which prioritizes the order in which neighbors are searched. A hypersphere centered at $\mathbf{y}$ of radius the best yet NN distance is considered, and leaves with the largest intersection to it are considered first. The search is halted after some pre-determined number of neighbors E_{max} has been searched.

BBF is open to various improvements—Muja and Lowe have developed an approach which chooses a ‘best’ algorithm for the problem in hand [Muja and Lowe, 2009]. A software suite implementing this is publicly available at <http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN>.

9.2.3 Classifier setting

A classifier based on discrimination functions is a deterministic machine—one pattern $\mathbf{x}$ will always be classified into the same class. Note however that one pattern may represent objects from different classes, so the classifier decision may be correct for some objects and incorrect for others. The choice of *optimal* classifier will seek to minimize these incorrect judgements in some sense.

Suppose a particular classifier is characterized by a parameter vector $\mathbf{q}$. The value of the mean loss $J(\mathbf{q})$ of this classifier depends on the decision rule that is applied: $\omega = d(\mathbf{x}, \mathbf{q})$. We will seek $\mathbf{q}^*$ that minimizes the mean loss:

$$J(\mathbf{q}^*) = \min_{\mathbf{q}} J(\mathbf{q}) \quad d(\mathbf{x}, \mathbf{q}) \in D. \quad (9.12)$$

—then $\mathbf{q}^*$ is the vector of optimal parameters.

The **minimum error criterion** (Bayes criterion, maximum likelihood) uses loss functions of the form $\lambda(\omega_r|\omega_s)$, where $\lambda(\cdot)$ describes quantitatively the loss incurred if a pattern $\mathbf{x}$ which should be classified into the class ω_s is incorrectly classified into the class ω_r .

$$\omega_r = d(\mathbf{x}, \mathbf{q}). \quad (9.13)$$

The mean loss is

$$J(\mathbf{q}) = \int_X \sum_{s=1}^R \lambda(d(\mathbf{x}, \mathbf{q})|\omega_s) p(\mathbf{x}|\omega_s) P(\omega_s) d\mathbf{x}, \quad (9.14)$$

where $P(\omega_s)$, $s = 1, \dots, R$ are the a priori probabilities of classes, and $p(\mathbf{x}|\omega_s)$, $s = 1, \dots, R$ are the conditional probability densities of objects $\mathbf{x}$ in the class ω_s .

A classifier that has been set according to the minimum-loss optimality criterion is easy to construct using discrimination functions; usually, unit loss functions are considered

$$\begin{aligned} \lambda(\omega_r|\omega_s) &= 0 \quad \text{for } r = s, \\ &= 1 \quad \text{for } r \neq s, \end{aligned} \quad (9.15)$$

and the discrimination functions are

$$g_r(\mathbf{x}) = p(\mathbf{x}|\omega_r) P(\omega_r) \quad r = 1, \dots, R, \quad (9.16)$$

where $g_r(\mathbf{x})$ corresponds (up to a multiplicative constant) to the value of the a posteriori probability $P(\omega_r|\mathbf{x})$.

This probability describes how often a pattern $\mathbf{x}$ is from the class ω_r . Clearly, the optimal decision is to classify a pattern $\mathbf{x}$ to a class ω_r if the a posteriori probability $P(\omega_r|\mathbf{x})$ is the highest of all possible a posteriori probabilities

$$P(\omega_r|\mathbf{x}) = \max_{s=1, \dots, R} P(\omega_s|\mathbf{x}). \quad (9.17)$$

Bayes theorem tells us

$$P(\omega_s|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_s) P(\omega_s)}{p(\mathbf{x})}, \quad (9.18)$$

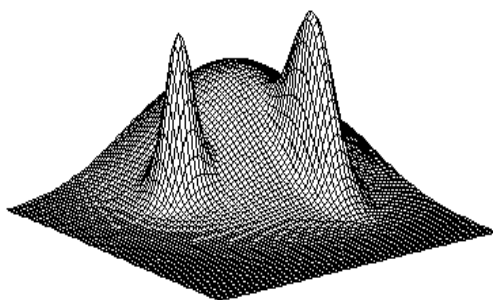


Figure 9.7: Minimum error classifier: a posteriori probabilities.
© Cengage Learning 2015.

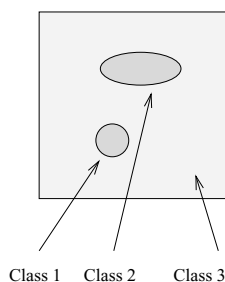


Figure 9.8: Minimum error classifier: discrimination hyper-surfaces and resulting classes. © Cengage Learning 2015.

where $p(\mathbf{x})$ is the mixture density. An example plot of a posteriori probabilities is shown in Figure 9.7, and corresponding discrimination hyper-surfaces for a three-class classifier are in Figure 9.8.

Analytic minimization of the extrema problem is in many practical cases impossible because the multi-dimensional probability densities are not available. Criteria for loss function evaluation can be found in [Sklansky, 1981; Devijver and Kittler, 1982]. The requirements for classification correctness, and the set of objects accompanied by information about their classes, are usually available in practical applications—very often this is all the information that can be used for the classifier design and setting.

The ability to set classification parameters from a set of examples is the very important issue of **classifier learning**. Classifier learning is based on a set of objects (represented by their feature vectors), each being accompanied by knowledge of its proper classification—this set of patterns and classes is called the **training set**. Clearly, the quality of the classifier setting depends on the quality and size of this finite training set. After setting, we may expect patterns that were not used for classifier learning to enter the classifier. Classifier learning methods must be **inductive** in the sense that information obtained from the elements of the training set must be generalized to cover the whole feature space, implying that the learning should be (near) optimal for all feasible patterns, not only for those patterns that were present in the training set. The classifier should be able to recognize even those objects that it has never ‘seen’ before.

It may be that a solution for a given problem does not exist. The larger the training set, the better the guarantee that the classifier may be set correctly—classification correctness and the size of the training set are closely related. If the statistical properties of patterns are known, the necessary sizes of the training sets can be estimated, but in reality this is rarely the case. The training set is actually supposed to substitute this

missing statistical information. Only after processing of the training set can the designer know whether it was sufficient, and whether an increase in size is necessary. Typically, training set size will be increased several times until the correct classification setting is achieved.

In practice, some proportion of the known training data will be held back to provide an objective test of performance when the classifier has been trained—this is called the test set, and will generate a measure of performance on hitherto unseen data. If just one data item is held back, this is called *leave one out testing*, and *leave K out testing* is defined similarly. If the K data items used for testing are selected randomly N times, a statistically robust testing scheme called *N -fold leave K out testing* results.

The general idea of sequential increase in training set size can be understood as presenting small portions of a large training set to the classifier, whose performance is checked after each portion—the smallest portion size is just one element. Sequential processing of information (which cannot be avoided in principle) has some substantial consequences in the classifier setting process.

All properties of classifier learning methods have analogies in the learning process of living organisms.

- **Learning** is the process of automated system optimization based on the sequential presentation of examples.
- The **goal of learning** is to minimize the optimality criterion. The criterion may be represented by the mean loss caused by incorrect decisions.
- Finite training set size requires the **inductive** character of learning. The goal of learning must be achieved by generalizing the information from examples, before all feasible examples have been presented. The examples may be chosen at random.
- The unavoidable requirements of sequential information presentation and the finite size of system memory necessitate the **sequential character of learning**. Therefore, learning is not a one-step process, but rather a step-by-step process of improvement.

The learning process searches out the optimal classifier setting from examples. The classifier system is constructed as a universal machine that becomes optimal after processing the training set examples (supervised learning), meaning that it is not necessary to repeat the difficult optimal system design if a new application appears. Learning methods do not depend on the application—it is possible to use the same learning to train a medical diagnostics classifier as an object recognition classifier for robot vision.

The quality of classifier decisions is closely related to the quality and amount of information that is available. From this point of view, patterns should represent as complex a description as possible, but conversely, this would imply a large number of description features. Therefore, object description is always a trade-off between permissible classification error, the complexity of the classifier construction, and the time required for classification. This results in a question of how to choose the best features from a set of available features, and how to detect the features with the highest contribution to the recognition success. Methods of determination of **informativity** and **discriminativity** of measured features can be found in [Young and Calvert, 1974; Pudil et al., 1994].

9.2.4 Classifier learning

Two common learning strategies will be presented in this section:

- **Probability density estimation** estimates the probability densities $p(\mathbf{x}|\omega_r)$ and probabilities $P(\omega_r)$, $r = 1, \dots, R$. The discrimination functions are computed according to the minimum error criterion (equation 9.16).
- **Direct loss minimization** finds the decision rule $\omega = d(\mathbf{x}, \mathbf{q}^*)$ by direct minimization of losses $J(\mathbf{q})$ without estimation of probability densities and probabilities. The criterion of the best approximation is applied.

Probability density estimation methods differ in computational difficulty according to the amount of prior information available about them. If some prior information is available, it usually describes the shape of probability density functions $p(\mathbf{x}|\omega_r)$. The parameters of the distribution are usually not known, and learning must estimate them. Therefore, this class of learning methods is sometimes called **parametric learning**.

Assume the patterns in the r^{th} class can be described by a normal distribution, $N(\boldsymbol{\mu}_r, \boldsymbol{\Psi}_r)$. The probability density for patterns from the class ω_r can be computed as

$$p(\mathbf{x}|\omega_r) = \frac{1}{(2\pi)^{n/2} \sqrt{\det \boldsymbol{\Psi}_r}} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_r)^T \boldsymbol{\Psi}_r^{-1} (\mathbf{x} - \boldsymbol{\mu}_r) \right), \quad (9.19)$$

where $\boldsymbol{\Psi}_r$ is the covariance matrix. Details about multi-variate probability density function estimation may be found in [Rao, 1965; Johnson and Wichern, 1990]. The computation process depends on additional information about $\boldsymbol{\mu}_r$ and $\boldsymbol{\Psi}_r$; three cases can be distinguished:

1. The covariance matrix $\boldsymbol{\Psi}_r$ is known, but the mean vector $\boldsymbol{\mu}_r$ is not. A feasible estimate may be the average

$$\tilde{\boldsymbol{\mu}}_r = \bar{\mathbf{x}} \quad (9.20)$$

which can be computed recursively

$$\bar{\mathbf{x}}(k+1) = \frac{1}{k+1} \left(k \bar{\mathbf{x}}(k) + \mathbf{x}_{k+1} \right), \quad (9.21)$$

where $\mathbf{x}_{k+1}$ is the $(k+1)^{st}$ pattern from the class r from the training set. This estimate is unbiased, consistent, efficient, and linear.

Alternatively, if the a priori estimate of the mean $\tilde{\boldsymbol{\mu}}_r(0)$ is available, the Bayes approach to estimation of the normal distribution parameters can be used. Then, the estimate can be computed recursively

$$\tilde{\boldsymbol{\mu}}_r(k+1) = \frac{a+k}{a+k+1} \tilde{\boldsymbol{\mu}}_r(k) + \frac{1}{a+k+1} \mathbf{x}_{k+1}. \quad (9.22)$$

The parameter a represents the confidence in the a priori estimate $\tilde{\boldsymbol{\mu}}_r(0)$. In training, a specifies the number of steps during which the designer believes more in the a priori estimate than in the mean value so far determined by training. Note that for $a = 0$, the Bayes estimate is identical to equation (9.21).

2. The covariance matrix Ψ_r is unknown, but the mean value vector μ_r is known. The estimate of the dispersion matrix Ψ_r is then usually taken as

$$\tilde{\Psi}_r = \frac{1}{K} \sum_{k=1}^K (\mathbf{x}_k - \mu_r)(\mathbf{x}_k - \mu_r)^T \quad (9.23)$$

or, in recursive form

$$\tilde{\Psi}_r(k+1) = \frac{1}{k+1} \left(k \tilde{\Psi}_r(k) + (\mathbf{x}_{k+1} - \mu_r)(\mathbf{x}_{k+1} - \mu_r)^T \right). \quad (9.24)$$

This estimate is unbiased and consistent.

As another option, if the a priori estimate $\tilde{\Phi}_r(0)$ of the dispersion matrix Ψ_r is known, the Bayes approach can be applied. Let K be the number of samples in the training set, and $\tilde{\Psi}_r(K)$ be calculated as in equation (9.24). Then

$$\tilde{\Phi}_r(K) = \frac{b \tilde{\Phi}_r(0) + K \tilde{\Psi}_r(K)}{b + K} \quad (9.25)$$

and $\tilde{\Phi}_r(K)$ is considered the Bayes estimate of the dispersion matrix Ψ_r . Parameter b represents the confidence in the a priori estimate $\tilde{\Phi}_r(0)$.

3. Both the covariance matrix Ψ_r and the mean value vector μ_r are unknown. The following estimates can be used

$$\tilde{\mu}_r = \bar{\mathbf{x}}, \quad (9.26)$$

$$\tilde{\Psi}_r = \mathbf{S} = \frac{1}{K-1} \sum_{k=1}^K (\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^T \quad (9.27)$$

or, in the recursive form

$$\begin{aligned} \mathbf{S}(k+1) = \frac{1}{k} \Big((k-1)\mathbf{S}(k) + (\mathbf{x}_{k+1} - \bar{\mathbf{x}}(k+1))(\mathbf{x}_{k+1} - \bar{\mathbf{x}}(k+1))^T \\ + k(\bar{\mathbf{x}}(k) - \bar{\mathbf{x}}(k+1))(\bar{\mathbf{x}}(k) - \bar{\mathbf{x}}(k+1))^T \Big). \end{aligned} \quad (9.28)$$

Alternatively, if the a priori estimate $\tilde{\Phi}_r(0)$ of the dispersion matrix Ψ_r and the a priori estimate $\tilde{\nu}_r(0)$ of the mean vector for class r are known, the Bayes estimates can be determined as follows

$$\tilde{\nu}_r(K) = \frac{a \tilde{\mu}_r(0) + K \tilde{\mu}_r(K)}{a + K}, \quad (9.29)$$

where K is the number of samples in the training set and $\tilde{\mu}_r(K)$ is determined using equations (9.20) and (9.21). The dispersion matrix estimate is calculated as

$$\begin{aligned} \tilde{\Phi}_r(K) = \frac{b}{b+K} \tilde{\Phi}_r(0) + a \tilde{\nu}_r(0) \tilde{\nu}_r(0)^T + (K-1) \tilde{\Psi}_r(K) \\ + K \tilde{\mu}_r(K) \tilde{\mu}_r(K)^T - (a+K) \tilde{\nu}_r(K) \tilde{\nu}_r(K)^T, \end{aligned} \quad (9.30)$$

where $\tilde{\Psi}_r(K)$ is calculated as given in equation (9.24). Then, $\tilde{\nu}_r(K)$ and $\tilde{\Phi}_r(K)$ are considered the Bayes estimates of the mean vector and the dispersion matrix for class r , respectively. Again, parameters a, b represent the confidence in the a priori estimates of $\tilde{\Phi}_r(0)$ and $\tilde{\nu}_r(0)$.

The a priori probabilities of classes $P(\omega_r)$ are estimated as relative frequencies

$$P(\omega_r) = \frac{K_r}{K}, \quad (9.31)$$

where K is the total number of objects in the training set; K_r is the number of objects from the class r in the training set.

Algorithm 9.3: Learning and classification by estimating normal distribution probability densities

1. Learning: Compute estimates of the mean vector $\boldsymbol{\mu}_r$ and the covariance matrix $\boldsymbol{\Psi}_r$.
2. Compute estimates of the a priori probability densities $p(\mathbf{x}|\omega_r)$, equation (9.19).
3. Compute estimates of the a priori probabilities of classes, equation (9.31).
4. Classification: Classify all patterns into the class r if

$$\omega_r = \max_{i=1,\dots,s} \left(p(\mathbf{x}|\omega_i) P(\omega_i) \right)$$

(equations 9.16 and 9.6).

If no prior information is available (i.e., even the distribution type is not known), the computation is more complex. In such cases, if it is not necessary to use the minimum error criterion, it is advantageous to use a direct loss minimization method.

No probability densities or probabilities are estimated in the second group of methods based on direct minimization of losses. The minimization process can be compared to gradient optimization methods, but pure gradient methods cannot be used because of unknown probability densities, so the gradient cannot be evaluated. Nevertheless, the minimum can be found using methods of **stochastic approximations** that are discussed in [Sklansky, 1981].

The most important conclusion is that the learning algorithms can be represented by recursive formulae in both groups of learning methods and it is easy to implement them.

9.2.5 Support vector machines

The support vector machine [SVM] approach has proved powerful and popular. Optimal classification of a separable two-class problem is achieved by maximizing the width of the empty area (**margin**) between the two classes—this width is defined as the distance between the discrimination hyper-surfaces in n -dimensional feature space. Vectors from each class that are closest to the discriminating surface are called **support vectors**. The support vectors thus specify the discrimination function. Usually, many discriminating hyper-surfaces exist, and the ability to identify the *optimal* discriminating surface is a major strength of this approach and assists in dealing with the problem of over-fitting during training. From the theoretical viewpoint, it can be shown that such an optimal

hyper-surface has the lowest **capacity**—a requirement based on the statistical learning theory of Vapnik and Chervonenkis [Burges, 1998].

Consider first the two-class linearly separable case, with a set of n -dimensional feature vectors $\mathbf{x}$ and class identifiers ω associated with them, $\omega \in \{-1, 1\}$. For simplicity, assume that the individual feature values x_i are scaled so that $x_i \in [0, 1]$ to overcome uneven influence from features with different variance. Discrimination between the two classes is achieved by defining a separating hyper-plane

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \quad (9.32)$$

(compare equation 9.7). To maximize the margin, two parallel hyper-planes are defined

$$\mathbf{w} \cdot \mathbf{x} + b = 1, \quad \mathbf{w} \cdot \mathbf{x} + b = -1, \quad (9.33)$$

passing through the support vectors and having no training patterns between them. To guarantee that no training patterns are present between these two hyper-planes, for all $\mathbf{x}_i$ we must have

$$\omega_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1. \quad (9.34)$$

Suppose $\mathbf{x}^+$ is on the positive plane, and $\mathbf{x}^-$ is the closest point on the negative plane. Then $\mathbf{x}^+ - \mathbf{x}^-$ is normal to the plane(s) and so for some λ

$$\begin{aligned} \lambda \mathbf{w} &= \mathbf{x}^+ - \mathbf{x}^-, \\ \lambda \|\mathbf{w}\|^2 &= \mathbf{x}^+ \cdot \mathbf{w} - \mathbf{x}^- \cdot \mathbf{w}, \\ \lambda \|\mathbf{w}\|^2 &= 2 \end{aligned} \quad (9.35)$$

and hence $|\mathbf{x}^+ - \mathbf{x}^-| = (2/\|\mathbf{w}\|)$. To maximize the margin, we need to minimize $\|\mathbf{w}\|$ subject to the constraint given in equation (9.34), which is a quadratic programming optimization problem. Once optimized, the discrimination hyperplane is parallel to and positioned in the middle between these two parallel hyper-planes (Figure 9.9c).

Lagrangian theory is employed to reformulate the minimization problem avoiding inequality constraints, and thereby simplifying the optimization process. The Lagrangian function is defined as

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i \omega_i (\mathbf{w}_i \cdot \mathbf{x}_i + b) + \sum_{i=1}^N \alpha_i, \quad (9.36)$$

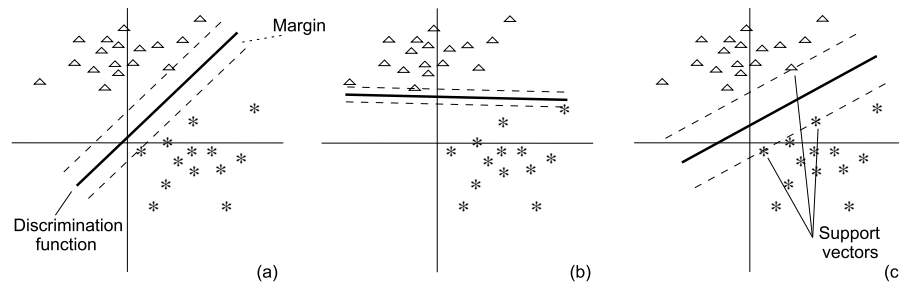


Figure 9.9: Basic two-class classification idea of support vector machines. (a) and (b) show two examples of non-optimal linear discrimination. (c) An optimal linear discriminator maximizes the margin between patterns of the two classes. The optimal hyperplane is a function of the support vectors. © Cengage Learning 2015.

where α_i are Lagrange multipliers. $L(\mathbf{w}, b, \alpha)$ is minimized with respect to $\mathbf{w}$ and b , while α_i are constrained to $\alpha_i \geq 0$. Taking the partial derivative of the Lagrangian function with respect to $\mathbf{w}$ and b results in

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^N \alpha_i \omega_i \mathbf{x}_i, \quad (9.37)$$

$$\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} = \sum_{i=1}^N \alpha_i \omega_i, \quad (9.38)$$

and setting both equations to zero creates the following relationships:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i \omega_i \mathbf{x}_i, \quad (9.39)$$

$$\sum_{i=1}^N \alpha_i \omega_i = 0. \quad (9.40)$$

These relations are substituted back into the original Lagrangian function, equation (9.36), to create an alternative optimization problem known as a dual formulation, taking into consideration pairwise relationships between training patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ and their respective class labels ω_i, ω_j ,

$$L(\mathbf{w}, b, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j \omega_i \omega_j (\mathbf{x}_i \cdot \mathbf{x}_j), \quad (9.41)$$

where $L(\mathbf{w}, b, \alpha)$ is maximized with respect to α_i , subject to $\sum_{i=1}^N \omega_i \alpha_i = 0$ and $\alpha_i \geq 0$. The scalar b has dropped out, but can be easily computed from the original dataset once equation (9.41) is optimized.

The two-class classification of a pattern $\mathbf{x}$ can be accomplished according to the following discrimination function:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}_i + b, \quad (9.42)$$

where

$$\begin{aligned} \omega_{\mathbf{x}} &= +1 & \text{if } f(\mathbf{x}) \geq 0, \\ &= -1 & \text{if } f(\mathbf{x}) < 0. \end{aligned} \quad (9.43)$$

The discrimination function is restated in terms of training vectors and multipliers by substituting $\mathbf{w}$ using equation (9.39)

$$f(\mathbf{x}) = \sum_{i \in \text{SV}} \alpha_i \omega_i (\mathbf{x}_i \cdot \mathbf{x}) + b. \quad (9.44)$$

Each of the Lagrange multipliers α_i shares a corresponding training vector $\mathbf{x}_i$. Those vectors that contribute to the maximized margin will have non-zero α_i and are the *support vectors*. The rest of the training vectors do not contribute to the final discrimination

function—therefore, the summation is only performed over those values of i for which $\mathbf{x}_i$ is a support vector ($i \in \text{SV}$).

Equation (9.43) shows that the decision as to whether a vector $\mathbf{x}$ belongs to class $\omega_{\mathbf{x}} = +1$ or $\omega_{\mathbf{x}} = -1$ depends only on the support vectors associated with the maximum margin as identified in the training phase. Therefore, the discrimination hyperplane in the feature space can be obtained from vectors in the input space and dot products in the feature space. Consequently, the training can be based on a small set of support vectors, even in large training sets, thus limiting the computational complexity of training with explicitly represented feature vectors.

If a separating hyperplane cannot be found to partition the feature space into two classes, one explanation is a linear inseparability of the training patterns. One solution is to tolerate some minimal mis-classification, but this is difficult to formulate as a quadratic programming problem. More commonly, the **kernel trick** (due to Aizerman [Aizerman et al., 1964]) is employed [Boser et al., 1992]. In the linear space, we determined the similarity of two patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ with a **kernel function** $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)$. The dot products of the linear support vector classifier can be replaced with non-linear kernel functions

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j). \quad (9.45)$$

—the idea here is to transform the vectors into a space in which a linear discrimination hyperplane can be determined, corresponding to a non-linear hyper-surface in the original feature space (Figure 9.10, see also Section 9.2.1, and equations 9.8–9.10). The support vector classification algorithm is formally identical to that used for linear discrimination, while every dot product used earlier is replaced by a non-linear kernel function. Due to the kernel trick, the support vector classifier can locate the linearly separating hyperplane in the transformed space by defining an appropriate kernel function.

While calculating the right-hand side of equation (9.45) may be expensive in general, a number of simple kernels can be used to serve the purpose, for example homogeneous d^{th} order polynomials, equation (9.46); non-homogeneous d^{th} order polynomials, equation (9.47); radial basis functions, equation (9.48); Gaussian radial basis functions,

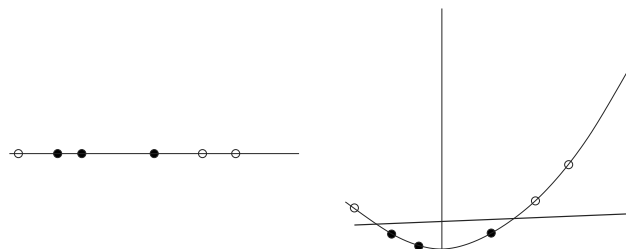


Figure 9.10: Achieving linear separability by application of a kernel function. On the left, the two classes are not linearly separable in 1D; on the right, the function $\Phi(x) = x^2$ creates a linearly separable problem. © Cengage Learning 2015.

equation (9.49); and other kernels:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d, \quad (9.46)$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d, \quad (9.47)$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \quad \text{for } \gamma > 0, \quad (9.48)$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{2\sigma^2}\right). \quad (9.49)$$

Applying the kernel technique to the dual Lagrange function yields

$$L(\mathbf{w}, b, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j \omega_i \omega_j k(\mathbf{x}_i, \mathbf{x}_j) \quad (9.50)$$

and the resulting discrimination function is

$$f(\mathbf{x}) = \sum_{i \in \text{sv}} \alpha_i \omega_i k(\mathbf{x}_i, \mathbf{x}) + b. \quad (9.51)$$

Linear support vector classification is identical to a classifier with linear discriminant functions if we somehow knew which training patterns constitute the support vectors, and disregard the remaining training patterns.

Algorithm 9.4 summarizes the training and classification steps using a support vector machine.

Algorithm 9.4: Support vector machine learning and classification

1. Training: Select an appropriate kernel function, $k(\mathbf{x}_i, \mathbf{x}_j)$.
2. Minimize $\|\mathbf{w}\|$ subject to the constraint given in equation (9.34). This is accomplished by maximizing the kernel-modified Lagrange function, equation (9.50), with respect to α_i subject to the constraints $\sum_{i=1}^N \alpha_i \omega_i = 0$ and $\alpha_i \geq 0$.
3. Store only the non-zero α_i and corresponding training vectors $\mathbf{x}_i$. These are the support vectors.
4. Classification: For each pattern $\mathbf{x}$, compute the discrimination function, equation (9.51) using the support vectors $\mathbf{x}_i$ and the corresponding weights α_i . The sign of the function determines the classification of $\mathbf{x}$.

Interestingly, the choice of the kernel function is frequently not critical and classifier performance is often comparable for several of the simple kernel functions, with performance much more dependent on data preparation [Schoelkopf, 1998; Hearst, 1998; Leopold and Kindermann, 2002].

Figure 9.11 shows an example of support vector machine training for a non-linearly separable problem; here, the kernel of equation (9.49) was used.

N -class classification is accomplished by combining N 2-class classifiers, each discriminating between a specific class and the rest of the training set. During the classification

stage, a pattern is assigned to the class with the largest positive distance between the classified pattern and the individual separating hyperplane for the N binary classifiers. (This approach can lead to very imbalanced classes in training, with one much more populous than the other—this issue is considered in [Li et al., 2009]). Alternatively, we might build $\frac{N(N-1)}{2}$ pairwise classifiers: classification is performed by applying them all and using a voting mechanism.

The support vector optimization problem can be solved analytically for small amounts of data, or if the support vector sets can be identified beforehand, but in most real-world cases the quadratic optimization problem must be solved numerically. For small problems, almost any general purpose quadratic optimization package will work sufficiently well. A survey of available solvers for larger-scale problems can be found in [More and Toraldo, 1993]. There is a substantial literature available discussing theoretical treatment as well as practical issues of support vector machine applications.

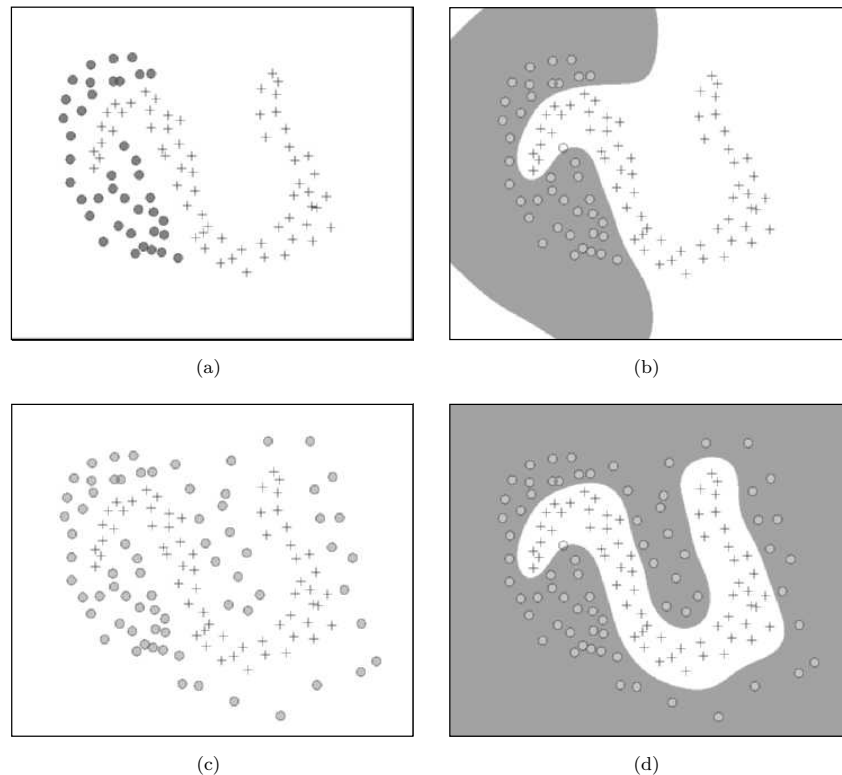


Figure 9.11: Support vector machine training: Gaussian radial basis function kernel used (equation 9.49). (a,c) Two-class pattern distribution in a feature space. (Note that the “+” patterns in (a) and (b) are identical while the “o” patterns in (a) are a subset of patterns in (b)). (b,d) Non-linear discrimination functions obtained after support vector machine training. © Cengage Learning 2015.

9.2.6 Cluster analysis

We noted earlier that unsupervised learning classification methods do not require training sets, and in particular do not need information about the class of objects in the learning stage. One such group of classification methods is called **cluster analysis**. Cluster analysis methods divide the set of patterns into subsets (clusters) based on the mutual similarity of subset elements. Objects that are dissimilar reside in different clusters.

Cluster analysis methods are *hierarchical* or *non-hierarchical*. Hierarchical methods construct a clustering tree; the set of patterns is divided into the two most dissimilar subsets, and subsets are recursively divided into smaller subsets. Non-hierarchical methods sequentially assign each pattern to one cluster. Methods and algorithms for cluster analysis can be found in [Duda et al., 2000; Everitt et al., 2001; Romesburg, 2004]

Non-hierarchical cluster analysis methods are either parametric or non-parametric. Parametric approaches are based on known class-conditioned distributions and require distribution parameter estimation similar to that used in minimum error classification described in Section 9.2.4.

Non-parametric cluster analysis is a popular, simple, and practically useful non-hierarchical approach to cluster analysis: the k -means algorithm is very well known and widely applied. It takes as input N n -dimensional data points and assumes the number of clusters K is known. The location of cluster *exemplars* is initialized, perhaps at random, or exploiting known data structure if available. Iteratively, data points are allocated to their closest exemplar, and then the exemplars are recomputed as the centroid of their associated data points. The algorithm terminates when the exemplars' positions have stabilized.

Algorithm 9.5: K -means cluster analysis

1. Define the number of clusters K to be sought in a set $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ of n -dimensional data.
2. Initialize the cluster starting points (exemplars, initial guesses) $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_K$, $\dots, \mathbf{v}_K$. This may be random, or from K data points, or from other prior knowledge.
3. Allocate data points to the closest $\mathbf{v}_i$ using some distance metric d (Euclidean is obvious and common).
4. Recompute the $\mathbf{v}_i$ as centroids of their associated data, V_i

$$V_i = \{\mathbf{x}_j : d(\mathbf{x}_j, \mathbf{v}_i) = \min_k(d(\mathbf{x}_j, \mathbf{v}_k))\} .$$
5. If the $\mathbf{v}_i$ have not stabilized, go to 3.

The 'error' in the clustering is given by the sum of square distances of the data points to their allocated exemplar

$$E = \sum_{i=1}^K \sum_{\mathbf{x}_j \in V_i} d^2(\mathbf{x}_j, \mathbf{v}_i) . \quad (9.52)$$

It is straightforward to show that the algorithm monotonically decreases this quantity (so it must converge).

K-means is exceptionally widely used as it is effective and simple. Minimizing the square error can be made computationally efficient and works well for compact hyper-spherical clusters. When Mahalanobis distance is used, K-means forms hyper-elliptical clusters.

Determining a good value for k automatically is a widely considered problem. A popular heuristic approach is to consider when the overall cost reduction by adding another cluster appears to benefit little: the graph of optimal E (equation 9.52) against K characteristically adopts an L-shape, and the location of the elbow of the L is a good choice for K . One approach locates the position of this plot that gives the best straight line fit to the left and right of each candidate K [Salvador and Chan, 2004]. A better theoretically founded approach considers incrementally increasing K by splitting the cluster which gives the optimal increase in the Bayesian Information Criterion—informally, splitting the ‘sparsest’ cluster—and then rerunning the algorithm [Pelleg and Moore, 2000]. Further answers to these and many related questions can be found in [Romesburg, 2004].

K-means is one of a family of algorithms that see wide use: similar others include the *ISODATA* cluster analysis method [Kaufman and Rousseeuw, 1990], the *mean shift* algorithm described in Section 7.1, *fuzzy k-means* (or *fuzzy c-means*) in which cluster membership is ‘fuzzy’ (see Section 9.7), *Kohonen networks* described in Section 9.3.2, *Hopfield networks* described in Section 9.3.3, and *Gaussian Mixture Models* described in Section 10.13. Note that statistical pattern recognition and cluster analysis can be combined—for instance, the minimum distance classifier can be taught using cluster analysis methods.

9.3 Neural nets

Neural nets have seen widespread use for some decades and represent a tool of great value in various areas generally regarded as ‘difficult’, particularly speech and visual pattern recognition.

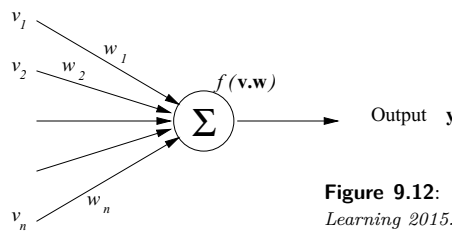


Figure 9.12: A simple (McCulloch-Pitts) neuron. © Cengage Learning 2015.

Most neural approaches are based on combinations of elementary processors (neurons), each of which takes a number of inputs and generates a single output. Associated with each input is a weight, and the output (in most cases) is then a function of the weighted sum of inputs; this output function may be discrete or continuous, depending on the variety of network in use. A simple neuron is shown in Figure 9.12—this model is derived from pioneering work on neural simulation conducted over 70 years ago [McCul-

loch and Pitts, 1943]. The inputs are denoted by $v_1, v_2, \dots$, and the weights by $w_1, w_2, \dots$; the total input to the neuron is then

$$x = \sum_{i=1}^n v_i w_i \quad (9.53)$$

or, more generally

$$x = \sum_{i=1}^n v_i w_i - \theta, \quad (9.54)$$

where θ is a threshold associated with this neuron. Also associated with the neuron is a **transfer function** $f(x)$ which provides the output; common examples are

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}, \quad (9.55)$$

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (9.56)$$

[Rosenblatt, 1962].

The general idea of collections (networks) of these neurons is that they are interconnected (so the output of one becomes the input of another, or others)—this idea mimics the high level of interconnection of elementary neurons found in brains, which is thought to explain the damage resistance and recall capabilities of humans. Such an interconnection may then take some number of external inputs and deliver up some (possibly different) number of external outputs—see Figure 9.13. What lies between then specifies the network: This may mean a large number of heavily interconnected neurons, or some highly structured (e.g., layered) interconnection, or, pathologically, nothing (so that inputs are connected straight to outputs).

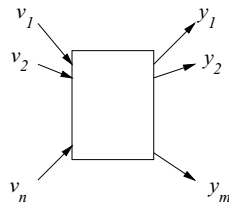


Figure 9.13: A neural network as a vector associator. © *Cengage Learning* 2015.

There are many uses to which such a structure may be put; the general task being performed is vector association. Examples may be:

- **Classification:** If the output vector (m -dimensional) is binary and contains only a single one, the position of the one classifies the input pattern into one of m categories.
- **Auto-association:** Some uses of neural networks cause them to re-generate the input pattern at the outputs (so $m = n$ and $v_i = y_i$); the purpose of this may be to derive a more compact vector representation from within the network internals.

- General association: At their most interesting, the vectors $\mathbf{v}$ and $\mathbf{y}$ represent patterns in different domains, and the network is forming a correspondence between them. For example, the inputs may represent a stream of written text and the outputs phonemes, making the network a speech generator [Sejnowski and Rosenberg, 1987].

Here we give only a brief overview to show the main principles of neural nets and their connections to conventional statistical pattern recognition. We do not discuss many alternative neural net techniques, methods, and implementations. Many further references may be found in a host of published papers and various introductory texts.

9.3.1 Feed-forward networks

Early neural networks involved no ‘internals’ (so the box in Figure 9.13 was empty); these *perceptrons* had a training algorithm developed which was shown to converge *if a solution to the problem at hand exists* [Minsky, 1988]; unfortunately, this proved very restrictive, requiring that the classification being performed be linearly separable. This restriction was overcome by widely used **back-propagation** algorithm [Rumelhart and McClelland, 1986], which trains strictly layered networks in which it is assumed that at least one layer exists between input and output (in fact, it can be shown that two such ‘hidden’ layers always suffice [Kolmogorov, 1963; Hecht-Nielsen, 1987]). Such a network is shown in Figure 9.14, and is an example of a **feed-forward** network: data are admitted at the inputs and travel in one direction toward the outputs, at which the ‘answer’ may be read.

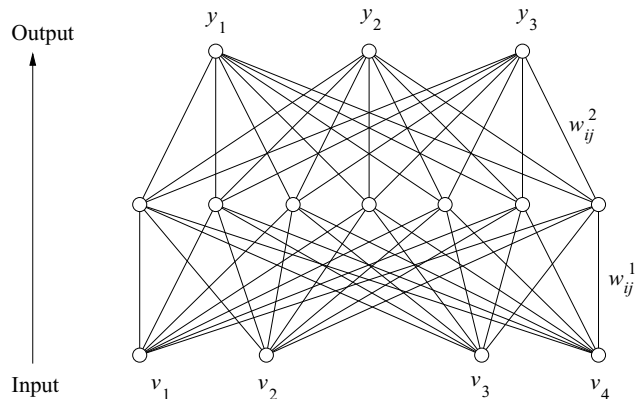


Figure 9.14: A three-layered neural net structure. © Cengage Learning 2015.

Such networks are ‘taught’ from a known training set. Back-propagation proceeds by comparing the output of the network to that expected, and computing an error measure based on sum of square differences. This is then minimized using gradient descent by altering the weights of the network. Denoting a member of the training set by $\mathbf{v}^i$, the actual outputs by $\mathbf{y}^i$, and the *desired* outputs by ω^i , the error is

$$E = \sum_i \sum_j (y_j^i - \omega_j^i)^2$$

and the algorithm performs the iterative updates

$$w_{ij}(k+1) = w_{ij}(k) - \epsilon \frac{\partial E}{\partial w_{ij}} \quad (9.57)$$

until ‘good’ performance is seen.

The literature on back-propagation is large and thorough, and we present here a summary of the algorithm only.

Algorithm 9.6: Back-propagation learning

1. Assign small random numbers to the weights w_{ij} , and set $k = 0$.
2. Input a pattern $\mathbf{v}$ from the training set and evaluate the neural net output $\mathbf{y}$.
3. If $\mathbf{y}$ does not match the required output vector ω , adjust the weights

$$w_{ij}(k+1) = w_{ij}(k) + \epsilon \delta_j z_i(k), \quad (9.58)$$

where ϵ is called the **learning constant** or **learning rate**, $z_i(k)$ is the output of the node i , k is the iteration number, δ_j is an error associated with the node j in the adjacent upper level

$$\delta_j = \begin{cases} y_j(1 - y_j)(\omega_j - y_j) & \text{for output node } j, \\ z_j(1 - z_j) \sum_l \delta_l w_{jl} & \text{for hidden node } j. \end{cases} \quad (9.59)$$

4. Go to step 2 and fetch the next input pattern.
5. Increment k , and repeat steps 2 to 4 until each training pattern outputs a suitably good approximation to that expected. Each circuit of this loop is termed an **epoch**.

The convergence process can be very slow, and there is an extensive literature on speeding the algorithm (see, for example, [Haykin, 1998]). The best known of these techniques is the introduction of **momentum**, which accelerates convergence across plateaus of the cost surface, and controls behavior in steep ravines. This approach rewrites equation (9.57) as

$$\Delta w_{ij} = \epsilon \frac{\partial E}{\partial w_{ij}}$$

and updates it to

$$\Delta w_{ij} := \epsilon \frac{\partial E}{\partial w_{ij}} + \epsilon \Delta w_{ij},$$

which updates equation (9.58) to

$$w_{ij}(k+1) = w_{ij}(k) + \epsilon \delta_j z_i(k) + \alpha (w_{ij}(k) - w_{ij}(k-1)); \quad (9.60)$$

α is called the **momentum constant** and is chosen to be between 0 and 1, having the effect of contributing a proportion of the update of the previous iteration into the current one. Thus, in areas of very low gradient, some movement continues.

9.3.2 Unsupervised learning

A different class of networks are self-teaching—that is, they do not depend on the net being exposed to a training set with known information about classes, but are able to self-organize themselves to recognize patterns automatically. Various types of networks exist under this general heading, of which the best known are Kohonen feature maps.

Kohonen maps take as input n -dimensional data vectors and generate an n -dimensional output that, within the domain of the problem at hand, ‘best represents’ the particular input given. More precisely, the network has a layer of neurons, each of which is connected to all n input vector components, each neuron calculates its input (equation 9.53), and that with the largest input is regarded as the ‘winner’; the n weights associated with the input arcs to this node then represent the output. Figure 9.15 illustrates this. The weights are updated using a learning algorithm that finds the data structure for itself (i.e., no prior classification is needed or indeed known). It may be clear that such a network is performing the role of clustering—similar inputs generate the same output.

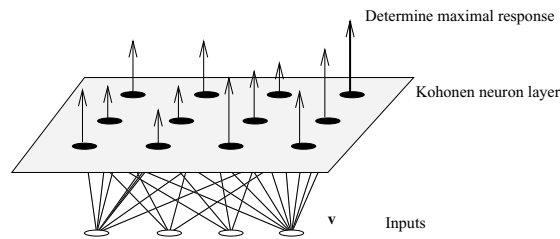


Figure 9.15: Kohonen self-organizing neural net. © Cengage Learning 2015.

The theory underlying Kohonen networks is derived from the operation of biological neurons, which are known to exist in locally 2D layers and in which neural responses are known to cluster. The derivation of the algorithm may be found in various standard texts [Kohonen, 1995] but may be summarized as follows.

Algorithm 9.7: Unsupervised learning of the Kohonen feature map

1. Assign random numbers with a small variance around the average values of the input feature vector elements to the weights w_{ij} .
2. Collect a sample of vectors $V = \{\mathbf{v}\}$ from the set to be analyzed.
3. Select a new vector $\mathbf{v} \in V$, and determine the neuron with the biggest input:

$$j^* = \operatorname{argmax}_j \sum_i w_{ij} v_i .$$

4. For all neurons n_j within a neighborhood of radius r of n_{j^*} , perform the weight update with step size $\alpha > 0$ (learning rate)

$$w_{ij} := w_{ij} + \alpha (v_i - w_{ij}) . \quad (9.61)$$

5. Go to step 3.
6. Reduce r and α , and go to step 3.

Several other varieties of self-teaching net exist, of which the best known is perhaps ART (Adaptive Resonance Theory) [Carpenter and Grossberg, 1987]. More specialized texts provide ample detail.

9.3.3 Hopfield neural nets

Hopfield nets are used mostly in optimization problems [Hopfield and Tank, 1986]; however, it is possible to represent recognition as an optimization task—find the maximum similarity between a pattern $\mathbf{x}$ and one of the existing exemplars $\mathbf{v}$.

In the Hopfield neural model, the network does not have designated inputs and outputs, but rather the current configuration represents its state. The neurons, which are fully interconnected, have discrete (0/1 or -1/1) outputs, calculated from equation (9.55). Weights between neurons do not evolve (learn), but are computed from a set of known exemplars at initialization

$$w_{ij} = \sum_r (v_i^r v_j^r), \quad i \neq j, \quad (9.62)$$

where w_{ij} is the interconnection weight between nodes i and j ; and v_i^r is the i^{th} element of the r^{th} exemplar; $w_{ii} = 0$ for any i .

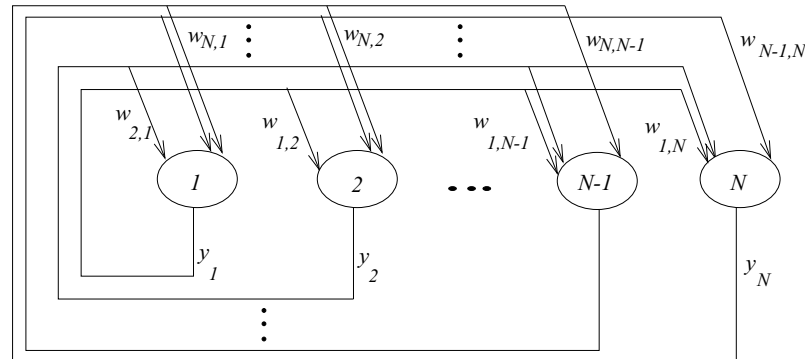


Figure 9.16: Hopfield recurrent neural net. © Cengage Learning 2015.

The Hopfield net acts as an associative memory where the exemplars are stored; its architecture is shown in Figure 9.16. When used for recognition, the feature vector to be classified enters the net in the form of initial values of node outputs. The Hopfield net then recurrently iterates using existing interconnections with fixed weights until a stable state is found—that such a state is reached can be proved under certain conditions (for which equation 9.62 is sufficient). The resulting stable state should be equal to the values of the exemplar that is closest to the processed feature vector in the Hamming metric sense. Supposing these class exemplars $\mathbf{v}^r$ are known, the recognition algorithm is as follows.

Algorithm 9.8: Recognition using a Hopfield net

1. Based on existing exemplars $\mathbf{v}^i$ of r classes, compute the interconnection weights w_{ij} (equation 9.62).
2. Apply an unknown feature vector $\mathbf{x}$ as initial outputs $\mathbf{y}(0)$ of the net.
3. Iterate until the net converges (output $\mathbf{y}$ does not change):

$$y_j(k+1) = f \sum_{i=1}^N w_{ij} y_i(k). \quad (9.63)$$

The final output vector $\mathbf{y}$ represents the exemplar of the class into which the processed feature vector $\mathbf{x}$ is classified. In other words, a Hopfield neural net transforms a non-ideal representation of an object (fuzzy, noisy, incomplete, etc.) to the ideal exemplar representation. The transformation of a noisy binary image of characters to a clear letter is one vision-related application.

The Hopfield neural net converges by seeking a minimum of a particular function—this is usually a *local* minimum, which may mean that the correct exemplar (the *global* minimum) is not found. Moreover, the number of local minima grows rapidly with the number of exemplars stored in the associative network. It can be shown that the minimum number of nodes N required is about seven times the number of memories M to be stored (this is known as the $0.15N \geq M$ rule) [McEliece et al., 1987], causing a rapid increase in the number of necessary nodes.

9.4 Syntactic pattern recognition

Quantitative description of objects using numeric parameters (the feature vector) is used in statistical pattern recognition, while **qualitative** descriptions are characteristic of syntactic pattern recognition. The object structure is contained in the syntactic description. If feature description is not able to represent the complexity of an object, or when an object can be represented as a hierarchical structure consisting of simpler parts, syntactic object description is appropriate. The elementary properties of the objects are called **primitives**: Section 8.2.4 covered the syntactic description of object borders using border primitives, these primitives representing parts of borders with a specific shape). Graphical or relational descriptions of objects where primitives represent subregions of specific shape is another example (see Sections 8.3.3 to 8.3.5). After each primitive has been assigned a symbol, relations between primitives in the object are described, and a **relational structure** results (Chapters 4 and 8). As in statistical recognition, the design of description primitives and their relation is not algorithmic. It is based on an analysis of the problem, designer experience, and abilities. However, there are some basic principles:

1. The number of primitive types should be small.
2. The primitives chosen must be able to form an appropriate object representation.

3. Primitives should be easily segmentable from the image.
4. Primitives should be easily recognizable using some statistical pattern recognition method.
5. Primitives should correspond with significant natural elements of the object (image) structure being described.

For example, if technical drawings are described, primitives are line and curve segments, binary relations describe relations such as *to be adjacent*, *to be left of*, *to be above*, etc. This description structure can be compared with the structure of a natural language. The text consists of sentences, sentences consist of words, words are constructed by concatenation of letters. Letters are considered primitives in this example; the set of all letters is called the **alphabet**. The set of all words in the alphabet that can be used to describe objects from one class (the set of all feasible descriptions) is named the **description language** and represents descriptions of all objects in the specific class. In addition, a **grammar** represents a set of rules that must be followed when words of the specific language are constructed from letters (of the alphabet). These definitions will be considered in more detail in Section 9.4.1.

Assume that the object is appropriately described by some primitives and their relations. Moreover, assume that the grammar is known for each class that generates descriptions of all objects of the specified class. Syntactic recognition decides whether the description word is or is not syntactically correct according to the particular class grammars, meaning that each class consists only of objects whose syntactic description can be generated by the particular grammar. Syntactic recognition is a process that looks for the grammar that can generate the syntactic word that describes an object.

Each relational structure with multiple relations can be transformed to a relational structure with at most binary relations; the image object is then represented by a **graph** which is **planar** if relations with adjacent regions only are considered. A graphical description is very natural, especially in the description of segmented images—examples were given in Section 8.3. Each planar graph can be represented either by a graph grammar or by a sequence of symbols (chain, word, etc.) over an alphabet. Sequential representation is not always advantageous in image object recognition because the valuable correspondence between the syntactic description and the object may be lost. Nevertheless, work with chain grammars is more straightforward and understandable, and all the main features of more complex grammars are included in chain grammars. Therefore, we will discuss principally sequential syntactic descriptions and chain grammars. More precise and detailed discussion of grammars, languages, and syntactic recognition methods can be found in [Fu, 1974, 1982].

Algorithm 9.9: Syntactic recognition

1. Learning: Based on the problem analysis, define the primitives and their possible relations.
2. Construct a description grammar for each class of objects using either hand analysis of syntactic descriptions or automated grammar inference (see Section 9.4.3).

3. Recognition: For each object, extract its primitives first; recognize the primitives' classes and describe the relations between them. Construct a description word representing an object.
4. Based on the results of the syntactic analysis of the description word, classify an object into that class for which its grammar (constructed in step 2) can generate the description word.

It can be seen that the main difference between statistical and syntactic recognition is in the learning process. Grammar construction can rarely be algorithmic using today's approaches, requiring significant human interaction. It is usually found that the more complex the primitives are, the simpler is the grammar, and the simpler and faster is the syntactic analysis. More complex description primitives on the other hand make step 3 of the algorithm more difficult and more time consuming; also, primitive extraction and evaluation of relations may not be simple.

9.4.1 Grammars and languages

Assuming that the primitives have been successfully extracted, all the inter-primitive relations can then be described syntactically as n -ary relations; these relations form structures (chains, trees, graphs) called **words** that represent the object or the pattern. Each pattern is therefore described by a word. Primitive classes can be understood as letters from the alphabet of symbols called **terminal symbols**. Let the alphabet of terminal symbols be V_t .

The set of patterns from a particular class corresponds to a set of words. This set of words is called the **formal language** and is described by a **grammar**. The grammar is a mathematical model of a generator of syntactically correct words (words from the particular language); it is a quadruple

$$G = [V_n, V_t, P, S], \quad (9.64)$$

where V_n and V_t are disjoint alphabets, elements of V_n are called **non-terminal symbols**, and elements of V_t are terminal symbols. Define V^* to be the set of all empty or non-empty words built from the terminal and/or non-terminal symbols. The symbol S is the grammar axiom or the *start* symbol. The set P is a non-empty finite subset of the set $V^* \times V^*$; elements of P are called the substitution rules. The set of all words that can be generated by the grammar G is called the **language** $L(G)$. Grammars that generate the same language are called **equivalent**.

A simple example will illustrate this terminology. Let the words generated by the grammar be squares of arbitrary size with sides parallel to the co-ordinate axes, and let the squares be represented by the Freeman chain code of the border in 4-connectivity (see Section 8.2.1). There are four terminal symbols (primitives) of the grammar in this case, $V_t = \{0, 1, 2, 3\}$. Let the non-terminal symbols be $V_n = \{s, a, b, c, d\}$. Note that the terminal symbols correspond to natural primitives of the 4-connectivity Freeman code; the non-terminal symbols were chosen from an infinite set of feasible symbols. The set of substitution rules P demonstrates how the start symbol $S = s$ can be transformed to

words corresponding to the Freeman chain code description of squares:

$$P : (1) s \rightarrow abcd, \quad (9.65)$$

$$(2) aAbBcCdD \rightarrow a1Ab2Bc3Cd0D, \quad (9.66)$$

$$(3) aAbBcCdD \rightarrow ABCD, \quad (9.67)$$

where A (B, C, D , respectively) is a variable representing any chain (including an empty one) consisting only of terminal symbols 1 (2, 3, 0). Rule 3 stops the word generating process. For example, a square with a side length $l = 2$ with the Freeman chain description 11223300 is generated by the following sequence of substitution rules (see Figure 9.17)

$$s \rightarrow^1 abcd \rightarrow^2 a1b2c3d0 \rightarrow^2 a11b22c33d00 \rightarrow^3 11223300,$$

where the arrow superscript refers to the appropriate substitution rule. The simple analysis of generated words shows that the language generated consists only of Freeman chain code representations of squares with sides parallel to the plane co-ordinates.

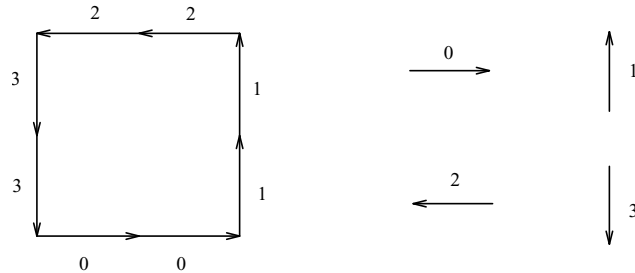


Figure 9.17: Square shape description. © Cengage Learning 2015.

Grammars can be divided into four main groups ordered from the general to the specific [Chomsky, 1966]:

1. Type 0—**General Grammars**

There are no limitations for the substitution rules.

2. Type 1—**Context-Sensitive Grammars**

Substitution rules can be of the form

$$W_1\alpha W_2 \rightarrow W_1UW_2, \quad (9.68)$$

which can contain the substitution rule $S \rightarrow e$, where e is an empty word; words W_1, W_2, U consist of elements of V^* , $U \neq e$, $\alpha \in V_n$. This means that the non-terminal symbols can be substituted by the word U in the context of words W_1 and W_2 .

3. Type 2—**Context-Free Grammars**

Substitution rules have the form

$$\alpha \rightarrow U, \quad (9.69)$$

where $U \in V^*$, $U \neq e$, $\alpha \in V_n$. Grammars can contain the rule $S \rightarrow e$. This means that the non-terminal symbol can be substituted by a word U independently of the context of α .

4. Type 3—**Regular Grammars**

The substitution rules of regular grammars are of the form

$$\alpha \rightarrow x\beta \quad \text{or} \quad \alpha \rightarrow x, \quad (9.70)$$

where $\alpha, \beta \in V_n$, $x \in V_t$. The substitution rule $S \rightarrow e$ may be included.

All the grammars discussed so far have been **non-deterministic**. The same left-hand side might appear in several substitution rules with different right-hand sides, and no rule exists that specifies which rule should be chosen. A non-deterministic grammar generates a language in which no words are ‘preferred’. If it is advantageous to generate some words (those more probable) more often than others, substitution rules can be accompanied by numbers (for instance, by probabilities) that specify how often the substitution rule should be applied. If the substitution rules are accompanied by probabilities, the grammar is called **stochastic**. If the accompanying numbers do not satisfy the properties of probability (unit sum of probabilities for all rules with the same left-hand side), the grammar is called **fuzzy** [Zimmermann et al., 1984].

Note that the evaluation of the frequency with which each substitution rule should be used can substantially increase the efficiency of syntactic analysis in the recognition stage [Fu, 1974].

9.4.2 Syntactic analysis, syntactic classifier

If appropriate grammars exist that can be used for representation of all patterns in their classes, the last step is to design a syntactic classifier which assigns the pattern (the word) to an appropriate class. It is obvious that the simplest way is to construct a separate grammar for each class; an unknown pattern x enters a parallel structure of blocks that can decide if $x \in L(G_j)$, where $j = 1, 2, \dots, R$ and R is the number of classes; $L(G_j)$ is the language generated by the j^{th} grammar. If the j^{th} block’s decision is positive, the pattern is accepted as a pattern from the j^{th} class and the classifier assigns the pattern to the class j . Note that generally more than one grammar can accept a pattern as belonging to its class.

The decision of whether or not the word can be generated by a particular grammar is made during **syntactic analysis**. Moreover, syntactic analysis can construct the pattern derivation tree which can represent the structural information about the pattern.

If a language is finite (and of a reasonable size), the syntactic classifier can search for a match between the word being analyzed and all the words of the language. Another simple syntactic classifier can be based on comparisons of the chain word descriptions with typical representatives of classes comparing primitive type presence only. This method is very fast and easily implemented, though it does not produce reliable results since the syntactic information is not used at all. However, impossible classes can be rejected in this step, which can speed up the syntactic analysis process.

Syntactic analysis is based on efforts to construct the tested pattern by the application of some appropriate sequence of substitution rules to the start symbol. If the substitution process is successful, the analysis process stops and the tested pattern can be generated by the grammar. The pattern can be classified into the class represented by the grammar. If the substitution process is unsuccessful, the pattern is not accepted as representing an object of this class.

If the class description grammar is regular (type 3), syntactic analysis is very simple. The grammar can be substituted with a finite non-deterministic automaton and it is easy to decide if the pattern word is accepted or rejected by the automaton [Fu, 1982]. If the grammar is context free (type 2), the syntactic analysis is more difficult. Nevertheless, it can be designed using stack automata.

Generally, which process of pattern word construction is chosen is not important; the transformation process can be done in top-down or bottom-up manner.

A top-down process begins with the start symbol and substitution rules are applied in the appropriate way to obtain the same pattern word as that under analysis. The final goal of syntactic analysis is to generate the same word as the analyzed word; every partial substitution creates a set of subgoals, just as new branches are created in the generation tree. Effort is always devoted to fulfill the current subgoal. If the analysis is not successful in fulfilling the subgoal, it indicates an incorrect choice of the substitution rule somewhere in the previous substitutions, and back-tracking is invoked to get back to the nearest higher tree level (closer to the root), and to pick another applicable rule. The process of rule applications and back-tracking is repeated until the required pattern word results. If the whole generating process ends unsuccessfully, the grammar does not generate the word, and the analyzed pattern does not belong to the class.

This top-down process is a series of expansions starting with the start symbol S . A bottom-up process starts with the analyzed word, which is **reduced** by applying reverse substitutions, the final goal being to reduce the word to the start symbol S . The main principle of bottom-up analysis is to detect subwords in the analyzed word that match the pattern on the right-hand side of some substitution rule, then the reduction process substitutes the former right-hand side with the left-hand side of the rule in the analyzed word. The bottom-up method follows no subgoals; all the effort is devoted to obtaining a reduced and simplified word pattern until the start symbol is obtained. Again, if the process is not successful, the grammar does not generate the analyzed word.

The pure top-down approach is not very efficient, since too many incorrect paths are generated. The number of misleading paths can be decreased by application of consistency tests. For example, if the word starts with a non-terminal symbol I , only rules with the right-hand side starting with I should be considered. Many more consistency tests can be designed that take advantage of prior knowledge. This approach is also called **tree pruning** (Figure 9.18) [Nilsson, 1982].

Tree pruning is often used if an exhaustive search cannot be completed because the search effort would exceed any reasonable bounds. Note that pruning can mean that the final solution is not optimal or may not be found at all (especially if tree search is used to find the best path through the graph, Section 6.2.4). This depends on the quality of the a priori information that is applied during the pruning process.

There are two main principles for recovery from following a wrong path. The first one is represented by the back-tracking mechanism already mentioned, meaning that the generation of words returns to the nearest point in the tree where another substitution rule can be applied which has not yet been applied. This approach requires the ability to re-construct the former appearances of generated subwords and/or remove some branches of the derivation tree completely.

The second approach does not include back-tracking. All possible combinations of the substitution rules are applied in parallel and several generation trees are constructed simultaneously. If any tree succeeds in generating the analyzed word, the generation process ends. If any tree generation ends with a non-successful word, this tree is abandoned.

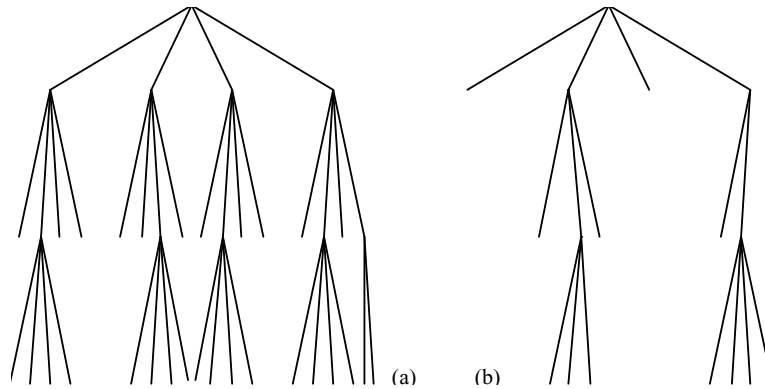


Figure 9.18: Tree pruning. (a) Original tree. (b) Pruning decreases size of the searched tree.
© Cengage Learning 2015.

The latter approach uses more brute force, but the algorithm is simplified by avoiding back-tracking.

It is difficult to compare the efficiency of these two and the choice depends on the application; Bottom-up analysis is more efficient for some grammars, and top-down is more efficient for others. As a general observation, the majority of syntactic analyzers which produce all generated words is based on the top-down principle. This approach is appropriate for most grammars but is usually less efficient.

Another approach to syntactic analysis uses example relational structures of classes. The syntactic analysis consists of matching the relational structure that represents the analyzed object with the example relational structure. The main goal is to find an **isomorphism** of both relational structures. These methods can be applied to n -ary relational structures as well. Relational structure matching is a perspective approach to syntactic recognition, a perspective way of image understanding (see Section 9.5). A simple example of relational structure matching is shown in Figure 9.19. A detailed description of relational structure matching approaches can be found in [Barrow and Popplestone, 1971; Ballard and Brown, 1982; Baird, 1984].

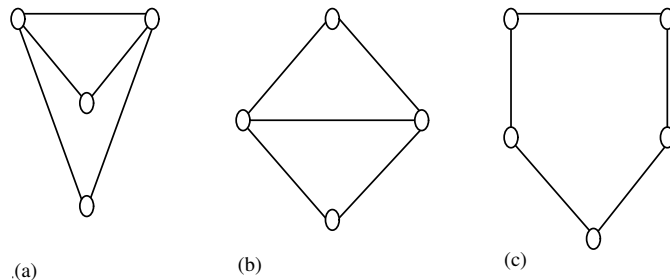


Figure 9.19: Matching relational structures: (a) and (b) match assuming nodes and relations of the same type; (c) does not match either (a) or (b). © Cengage Learning 2015.

9.4.3 Syntactic classifier learning, grammar inference

To model a language of any class of patterns as closely as possible, the grammar rules should be extracted from a training set of example words. This process of grammar construction from examples is known as **grammar inference**, the essence of which can be seen in Figure 9.20.

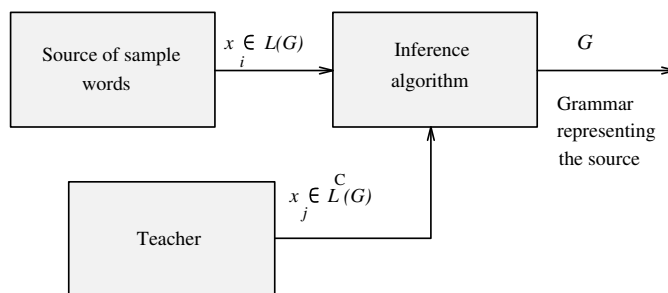


Figure 9.20: Grammar inference. © Cengage Learning 2015.

The source of words generates finite example words consisting of the terminal symbols. Assume that these examples include structural features that should be represented by a grammar G which will serve as a model of this source. All the words that can be generated by the source are included in the language $L(G)$, and the words that cannot be generated by the source represent a residuum of this set $L^C(G)$. This information enters the inference algorithm whose goal is to find and describe the grammar G . Words that are included in the language $L(G)$ can be acquired simply from the source of examples. However, the elements of $L^C(G)$ must be presented by a teacher that has additional information about the grammar properties [Barrero, 1991].

Note that the number of example words generated by the source is finite and it is therefore not sufficient to define the possibly infinite language $L(G)$ unambiguously. Any finite set of examples can be represented by an infinite set of languages, making it impossible to identify unambiguously the grammar that generated the examples. Grammar inference is expected to construct the grammar that describes the training set of examples, plus another set of words that in some sense have the same structure as the examples.

The inference methods can be divided into two groups, based on **enumeration** and **induction**. Enumeration detects the grammar G from the finite set M of grammars that can generate the entire training set of examples or its main part. The difficulty is in the definition of the set M of grammars and in the procedure to search for the grammar G . Induction-based methods start with the analysis of words from the training set; the substitution rules are derived from these examples using patterns of similar words.

There is no general method for grammar inference that constructs a grammar from a training set. Existing methods can be used to infer regular and context-free grammars, and may furthermore be successful in some other special cases. Even if simple grammars are considered, the inferred grammar usually generates a language that is much larger than the minimum language that can be used for appropriate representation of the class. This property of grammar inference is extremely unsuitable for syntactic analysis because of the computational complexity. Therefore, the main role in syntactic analyzer learning

is still left to a human analyst, and the grammar construction is based on heuristics, intuition, experience, and prior information about the problem.

If the recognition is based on sample relational structures, the main problem is in its automated construction. The conventional method for the sample relational structure construction is described in [Winston, 1975], where the relational descriptions of objects from the training set are used. The training set consists of examples and counter-examples. The counter-examples should be chosen to have only one typical difference in comparison with a pattern that is a representative of the class.

9.5 Recognition as graph matching

Graphs with weighted nodes and weighted arcs often appear in image descriptions using relational structures. Graph comparisons can tell us whether the reality represented by an image matches prior knowledge incorporated into graphical models. Figure 9.21 shows a typical graph matching task.

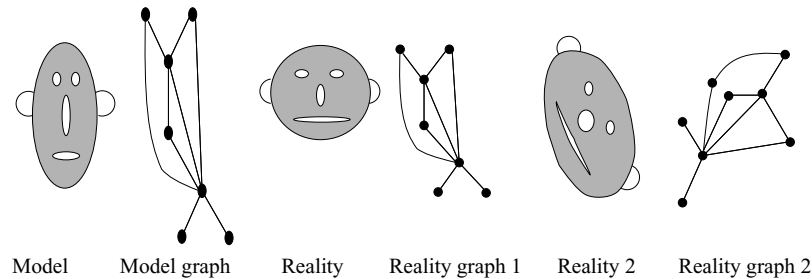


Figure 9.21: Graph matching problem. © Cengage Learning 2015.

If the problem is to find an object (represented by a model graph) in a graphical representation of the image, the model must match a subgraph in the image graph exactly. An exact match of graphs is called graph **isomorphism**—for example, the graphs in Figure 9.21 are isomorphic.

Graph isomorphism and subgraph isomorphism evaluation is a classical problem in graph theory and is important from both practical and theoretical points of view. Graph theory and algorithms have an extensive literature (for example, [Harary, 1969; McHugh, 1990; Tucker, 1995; Bollobas, 2002]), The problem is actually more complex in reality, since the requirement of an exact match is very often too strict in recognition problems.

Because of imprecise object descriptions, image noise, overlapping objects, lighting conditions, etc., the object graph usually does not match the model graph exactly. Graph matching is a difficult problem, and evaluation of graph **similarity** is not any easier—an important problem is to design a metric which determines how similar two graphs are.

9.5.1 Isomorphism of graphs and subgraphs

Graph isomorphism problems can be divided into three main classes [Harary, 1969; Ballard and Brown, 1982].

1. **Graph isomorphism.** Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, find a *one-to-one* and *onto* mapping (an isomorphism) f between V_1 and V_2 such that for each edge of E_1 connecting any pair of nodes $v, v' \in V_1$, there is an edge of E_2 connecting $f(v)$ and $f(v')$; further, if $f(v)$ and $f(v')$ are connected by an edge in G_2 , v and v' are connected in G_1 .
2. **Subgraph isomorphism.** Find an isomorphism between a graph G_1 and subgraphs of another graph G_2 . This problem is more difficult than the previous one.
3. **Double subgraph isomorphism.** Find all isomorphisms between subgraphs of a graph G_1 and subgraphs of another graph G_2 . This problem is of the same order of difficulty as number 2.

It is known that the subgraph and double subgraph isomorphism problems are *NP*-complete (using known algorithms, solution time is proportional to an exponential function of input length)—it is still not known whether the graph isomorphism problem is *NP*-complete. Non-deterministic algorithms for graph isomorphism that use heuristics and look for suboptimal solutions give a solution in polynomial time in both graph and subgraph isomorphism testing. Isomorphism testing is computationally expensive for both weighted and unweighted graphs.

Node and arc weights can simplify isomorphism testing. More precisely, the weights may make disproof of isomorphism easier. Isomorphic weighted graphs have the same number of nodes and arcs with the same weight. An isomorphism test of two weighted graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ can be based on partitioning the node sets V_1 and V_2 in a consistent manner looking for inconsistencies in the resulting set partitions—the goal is to achieve a one-to-one correspondence between nodes from sets V_1 and V_2 . The algorithm consists of repeated node set partitioning steps, and the necessary conditions of isomorphism are tested after each step (the same number of nodes of equivalent properties in corresponding sets of both graphs). The node set partitioning may, for example, be based on the following properties:

- Node attributes (weights).
- The number of adjacent nodes (connectivity).
- The number of edges of a node (node degree).
- Types of edges of a node.
- The number of edges leading from a node back to itself (node order).
- The attributes of adjacent nodes.

After new subsets are generated based on one of these criteria, the cardinalities of corresponding subsets of nodes in graphs G_1 and G_2 are tested; see Figure 9.22a. Obviously, if v_{1i} is in several subsets V_{1j} , then the corresponding node v_{2i} must also be in the corresponding subsets V_{2j} , or the isomorphism is disproved

$$v_{2i} \in \bigcap_{j|v_{1i} \in V_{1j}} V_{2j} . \quad (9.71)$$

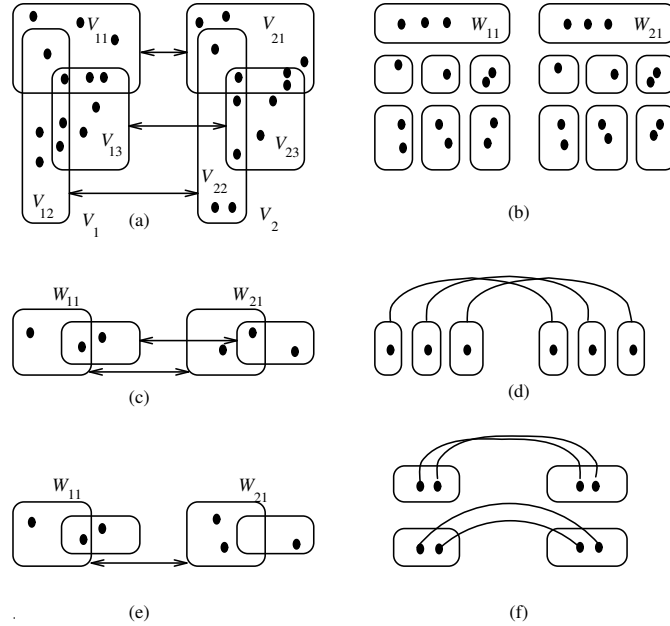


Figure 9.22: Graph isomorphism. (a) Testing cardinality in corresponding subsets. (b) Partitioning node subsets. (c) Generating new subsets. (d) Subset isomorphism found. (e) Graph isomorphism disproof. (f) Situation when arbitrary search is necessary. © Cengage Learning 2015.

If all the generated subsets satisfy the necessary conditions of isomorphism in step i , the subsets are split into new sets of nodes W_{1n}, W_{2n} (Figure 9.22b)

$$\begin{aligned} W_{1i} \cap W_{1j} &= \emptyset \quad \text{for } i \neq j, \\ W_{2i} \cap W_{2j} &= \emptyset \quad \text{for } i \neq j. \end{aligned} \quad (9.72)$$

Clearly, if $V_{1j} = V_{2j}$ and if $v_{1i} \notin V_{1k}$, then $v_{2i} \in V_{2k}^C$, where V^C is the set complement. Therefore, by equation (9.71), corresponding elements v_{1i}, v_{2i} of W_{1n}, W_{2n} must satisfy

$$v_{2i} \in \left\{ \bigcap_{\{j|v_{1i} \in W_{1j}\}} W_{2j} \right\} \cap \left\{ \bigcap_{\{(k|v_{1i} \notin W_{1k}) \wedge (W_{1k} = W_{2k})\}} W_{2k}^C \right\}. \quad (9.73)$$

The cardinality of all the corresponding sets W_{1n}, W_{2n} is tested to disprove the graph isomorphism.

The same process is repeated in the following steps, applying different criteria for node subset generation. Note that new subsets are generated independently in W_{1i}, W_{2i} (Figure 9.22c).

The process is repeated unless one of three cases occurs:

1. All the corresponding sets W_{1i}, W_{2i} contain one node each. The isomorphism is found (Figure 9.22d).

2. The cardinality condition is not satisfied in at least one of the corresponding subsets. The isomorphism is disproved (Figure 9.22e).
3. No more new subsets can be generated before one of the previous cases occurs. In that situation, either the node set partitioning criteria are not sufficient to establish an isomorphism, or more than one isomorphism is possible. In this case, systematic arbitrary assignment of nodes that have more than one possible corresponding node and cardinality testing after each assignment may provide a solution (Figure 9.22f).

The last part of the process, based on systematic assignment of possibly corresponding nodes and isomorphism testing after each assignment, may be based on back-tracking principles. Note that the back-tracking approach can be used from the very beginning but it is more efficient to start the test using all available prior information about the graphs. The back-tracking process is applied if more than one potential correspondence between nodes is encountered. Back-tracking tests for directed graph isomorphism and a recursive algorithm are given in [Ballard and Brown, 1982] together with accompanying hints for improving efficiency.

Algorithm 9.10: Graph isomorphism

1. Take two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$.
2. Use a node property criterion to generate subsets V_{1i}, V_{2i} of the node sets V_1 and V_2 . Test whether the cardinality conditions hold for corresponding subsets. If not, the isomorphism is disproved.
3. Partition the subsets V_{1i}, V_{2i} into subsets W_{1j}, W_{2j} satisfying the conditions given in equation (9.72) (no two subsets W_{1j} or W_{2j} contain the same node). Test whether the cardinality conditions hold for all the corresponding subsets W_{1j}, W_{2j} . If not, the isomorphism is disproved.
4. Repeat steps 2 and 3 using another node property criterion in all subsets W_{1j}, W_{2j} generated so far. Stop if one of the three above-mentioned situations occurs.
5. Based on the situation that stopped the repetition process, the isomorphism either was proved, disproved, or some additional procedures (such as back-tracking) must be applied to complete the proof or disproof.

A classic approach to subgraph isomorphism uses a brute force enumeration process, described as a depth-first tree search algorithm [Ullmann, 1976]. As a way of improving efficiency, a refinement procedure is entered after each node is searched in the tree—the procedure reduces the number of node successors, which yields a shorter execution time. An alternative approach transforms the graph problem into a linear programming problem [Zdrahal, 1981].

The double subgraph isomorphism problem can be translated into a subgraph isomorphism problem using the **clique**—a complete (totally connected) subgraph—approach. A clique is said to be maximal if no other clique properly includes it. Note that a graph may have more than one maximal clique; however, it is often important to find the largest

maximal clique: this is a well-known problem in graph theory. An example algorithm for finding all cliques of an undirected graph can be found in [Bron and Kerbosch, 1973]. The maximal clique $G_{\text{clique}} = (V_{\text{clique}}, E_{\text{clique}})$ of the graph $G = (V, E)$ can be found as follows [Niemann, 1990].

Algorithm 9.11: Maximal clique location

1. Take an arbitrary node $v_j \in V$; construct a subset $V_{\text{clique}} = \{v_j\}$.
2. In the set V_{clique}^C search for a node v_k that is connected with all nodes in V_{clique} . Add the node v_k to a set V_{clique} .
3. Repeat step 2 as long as new nodes v_k can be found.
4. If no new node v_k can be found, V_{clique} represents the node set of the maximal clique subgraph G_{clique} (the maximal clique that contains the node v_j).

To find the largest maximal clique, an additional maximizing search is necessary. Other clique-finding algorithms are discussed in [Ballard and Brown, 1982; Yang et al., 1989].

The search for isomorphism of two subgraphs (the double subgraph isomorphism) is transformed to a clique search using the **assignment graph** [Ambler, 1975]. A pair (v_1, v_2) , $v_1 \in V_1$, $v_2 \in V_2$ is called an **assignment** if the nodes v_1 and v_2 have the same node property descriptions, and two assignments (v_1, v_2) and (v'_1, v'_2) are **compatible** if (in addition) all relations between v_1 and v'_1 also hold for v_2 and v'_2 (graph arcs between v_1, v'_1 and v_2, v'_2 must have the same weight, including the no-edge case). The set of assignments defines the set of nodes V_a of the assignment graph G_a . Two nodes in V_a (two assignments) are connected by an arc in the assignment graph G_a if these two nodes are compatible. The search for the maximum matching subgraphs of graphs G_1 and G_2 is a search for the maximum totally connected subgraph in G_a (the maximum totally compatible subset of assignments).

The maximum totally connected subgraph is a maximal clique, and the maximal clique-finding algorithm can be applied to solve this problem.

9.5.2 Similarity of graphs

Perfect matches are unlikely in reality, and the foregoing approaches are not able to distinguish between a small mismatch of two very similar graphs and the case when the graphs are not similar at all. If graph similarity is sought, the main stress is given to the ability to quantify the similarity. Having three graphs G_1, G_2, G_3 , the question as to which two are more similar is a natural one [Buckley, 1990].

The similarity of two strings (chains) can be based on the **Levenshtein distance**, which is defined as the smallest number of deletions, insertions, and substitutions necessary to convert one string into the other [Schlesinger and Hlaváč, 2002]. Transformations of string elements can be assigned a specific transition cost to make the computed similarity (distance) more flexible and more sensitive. This principle can be applied to graph similarity as well. A set of feasible transformations of nodes and arcs (insertion, deletion, substitution, relabeling) is defined, and these transformations are accompanied by

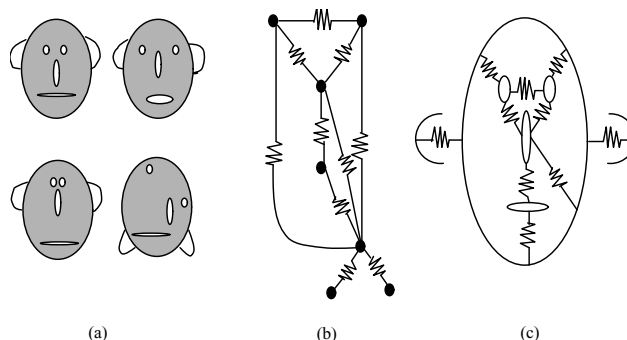


Figure 9.23: Templates and springs principle. (a) Different objects having the same description graphs. (b), (c) Nodes (templates) connected by springs, graph nodes may represent other graphs in finer resolution. © Cengage Learning 2015.

transition costs. Any sequence of transformations is assigned a combination of single step costs (like the sum of individual costs). The set of transformations that has the minimum cost and transforms one graph to another graph defines a distance between them [Niemann, 1990].

Note that similarity can be searched for in hierarchical graph structures. The graphs consist of a number of subgraphs in which isomorphism (or similarity) has already been proved. The next step is to detect, describe, and evaluate relations between these subgraphs (Figure 9.23, cf. Figure 9.21).

To explain the principles, a physical analogy of templates and springs [Fischler and Elschlager, 1973] is usually considered. The templates (subgraphs) are connected by springs (relations between subgraphs). The quality of the match of two graphs relates to the quality of the local fit (in corresponding templates) and to the amount of energy used to stretch the springs to match one graph onto the second (reference) graph. To make the graph similarity measure more flexible, extra costs may be added for missing parts of the graph as well as for some extra ones. The spring energy penalty may be made highly non-linear, better to reflect the descriptive character in particular applications.

9.6 Optimization techniques in recognition

Optimization is very flexible. In image recognition and understanding, the best image representation is sought, or the best match between image and model is required, or the best image understanding is the goal. Whenever ‘the best’ is considered, some objective function of *goodness* must be available, implying that an optimization technique can be applied which looks for the maximum ... for *the best*.

A function optimization problem is defined by some finite domain D and a function $f : D \rightarrow R$, R being the set of real numbers; optimization seeks the *best* value in D under f . The ‘best’ is understood as finding a value $\mathbf{x} \in D$ yielding either the minimum or the maximum of the function f :

$$f_{\min}(\mathbf{x}) = \min_{\mathbf{x} \in D} f(\mathbf{x}), \quad f_{\max}(\mathbf{x}) = \max_{\mathbf{x} \in D} f(\mathbf{x}). \quad (9.74)$$

The function f is called the **objective** function. Optimization methods for seeking maxima and minima are logically equivalent, and optimization techniques can be equally useful if either a maximum or minimum is required.

No optimization algorithm can guarantee finding a good solution to a problem if the objective function does not reflect the *goodness* of the solution. Therefore, the design of the objective function is a key factor in the performance of any optimization algorithm.

Most conventional approaches to optimization use calculus-based methods which can be compared to climbing a hill gradient of the objective function gives the steepest direction to climb. The main limitation of calculus-based methods is their local behavior; the search can easily end in a local maximum, and the global maximum can be missed (see Figure 9.24).



Figure 9.24: Limitations of hill climbing methods. © Cengage Learning 2015.

Several methods improve the probability of finding the global maximum; to start the hill climbing at several points in the search space, to apply enumerative searches such as dynamic programming, to apply random searches, etc. Among these possibilities are genetic algorithms and simulated annealing.

9.6.1 Genetic algorithms

Genetic algorithms (GA) use natural evolution mechanisms to search for the maximum of an objective function [Goldberg, 1989; Mitchell, 1998; Haupt and Haupt, 2004]. GAs do not guarantee that the global optimum will be found, but empirical results from many applications show that the final solution is usually very close to it. This is very important in image understanding applications, as will be seen in the next chapter. There are almost always several consistent (feasible) solutions that are locally optimal in image understanding, or matching, and only one of those possible solutions is the best one represented by the global maximum. The opportunity to find a (near) global optimum is very valuable in these tasks.

GAs differ substantially from other optimization methods in the following ways [Goldberg, 1989].

1. GAs work with a coding of the parameter set, not the parameters themselves. They require the natural parameter set of the problem to be coded as a finite-length string over some finite alphabet. This implies that any optimization problem representation must be transformed to a string representation; binary strings are

often used. The design of the problem representation as a string is an important part of the method.

2. GAs search from a population of points, not a single point. The population of solutions that is processed in each step is large, meaning that the search for the optimum is driven from many places in the search space simultaneously. This gives a better chance of finding the global optimum.
3. GAs use the objective function directly, not derivatives or other auxiliary knowledge. The search for new, better solutions depends on the values of the evaluation function only. Note that, as in other recognition methods, GAs find the (near) global optimum of the evaluation function but there is no guarantee at all that the evaluation function is relevant to the problem. The evaluation function describes the *goodness* of the particular string. The value of the evaluation function is called **fitness** in GAs.
4. GAs use probabilistic transition rules, not deterministic rules. Rules of transition from the current population of strings to a new and better population of strings are based on the natural idea of supporting good strings with higher fitness and removing poor strings with lower fitness. The best strings representing the best solutions are allowed to survive the evolution process with a higher probability.

The survival of the fittest and the death of poor code strings is achieved by applying three basic operations: **reproduction**, **crossover**, and **mutation**.

The population of strings represents all the strings that are being processed in the current step of the GA. The sequence of reproduction, crossover, and mutation generates a new population of strings from the previous population.

Reproduction

The reproduction operator is responsible for the survival of the fittest and for the death of others based on a probabilistic treatment.

The mechanism copies strings with highest fitness into the next generation of strings. The selection process is usually probabilistic, the probability that a string is reproduced into the new population being given by its relative fitness in the current population—this is their mechanism of survival. The lower the fitness of the string, the lower the chances for survival. This process results in a set of strings where some strings of higher fitness may be copied more than once into the next population. The total number of strings in the population usually remains unchanged, and the average fitness of the new generation is higher than it was before.

Crossover

There are many variations of crossover. The basic idea is to ‘mate’ newly reproduced strings at random, randomly choosing a position for the border of each pair of strings, and to produce new strings by swapping all characters between the beginning of the string pairs and the border position; see Figure 9.25.

There is a probability parameter representing the number of pairs which will be processed by crossover, so not all newly reproduced strings are subject to it. It may be performed such that the best reproduced strings are kept in an unchanged form.

Crossover together with reproduction represent the main power of GAs. However, there is one more idea in the crossover operation: Blocks of characters can be detected in strings that have locally correct structure even if the string as a whole does not represent a good solution. These blocks of characters in strings are called **schemata**: they are substrings that can represent building blocks of the string, and can be understood as the local pattern of characters. Clearly, if schemata can be manipulated as locally correct blocks, the optimal solution can be located faster than if all the characters are handled independently. In every generation of n strings, about n^3 schemata are processed. This is called the **implicit parallelism** of genetic algorithms [Goldberg, 1989].

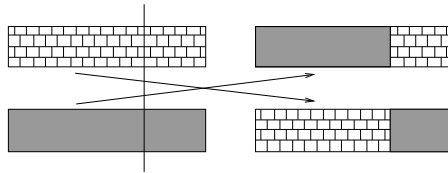


Figure 9.25: Principle of crossover. Two strings before (left) and after the crossover (right). © Cengage Learning 2015.

Mutation

Mutation plays only a secondary role in GAs. Its principle is randomly to change one character of some string of the population from time to time—it might, for example, take place approximately once per thousand bit transfers. The main reason for mutation is the fact that some local configurations of characters in strings of the population can be totally lost as a result of reproduction and crossover—mutation protects GAs against such irrecoverable loss of good solution features.

Population convergence in GAs is a serious question. For practical purposes this question becomes one of when to stop generating new string populations. A common and practically proven criterion recommends that the population-generating process be stopped when the maximum achieved fitness in the population has not improved substantially through several previous generations.

The starting population can be generated at random, assuming the alphabet of characters and the desired length of strings are known. Nevertheless, as always, if some prior knowledge about the solution is available (the probable local patterns of characters, the probable percentages of characters in strings, etc.), then it is advantageous to use this information to make the starting population as fit as possible. The better the starting population, the easier and faster the search for the optimum.

Algorithm 9.12: Genetic algorithm

1. Create a starting population of code strings, and find the value of their objective functions.
2. Probabilistically reproduce high fitness strings in the new population, remove poor fitness strings (reproduction).
3. Construct new strings combining reproduced code strings from the previous population (crossover).

4. From time to time, change one character of some string at random (mutation).
5. Order code strings of the current population according to the value of their objective function (fitness).
6. If the maximum achieved string fitness does not increase over several steps, stop—otherwise return to step 2. The desired optimum is represented by the current string of maximum fitness.

See Section 10.10.2 for an example of the algorithm use for image analysis. A more detailed and precise description of genetic algorithms can be found in [Goldberg, 1989; Rawlins, 1991; Mitchell, 1996]. Many examples and descriptions of related techniques are included there as well, such as knowledge implementation into mutation and crossover, GA learning systems, hybrid techniques that combine good properties of conventional hill climbing searches and GAs, etc.

9.6.2 Simulated annealing

Simulated annealing [Kirkpatrick et al., 1983] represents another group of robust optimization methods. Similarly to GAs, simulated annealing searches for an extremum of an objective function (cost function) that represents the fitness of some complex system. Searching for a *minimum* is considered in this section because it simplifies the energy-related metaphor with the natural behavior of matter. Simulated annealing combines two basic optimization principles, **divide and conquer** and **iterative improvement** (hill climbing). This combination avoids getting stuck in local optima. It has as a basis a strong connection with statistical mechanics or thermodynamics, and multi-variate or combinatorial optimization. Simulated annealing may be suitable for *NP*-complete optimization problems; it does not guarantee that the global optimum is found, but the solution is usually near-optimal.

To explain the principle of simulated annealing, imagine a sugar bowl filled with cube sugar [Cerny, 1985]. Usually, some cubes do not fit and the lid cannot be closed. From experience, everybody knows that shaking the bowl will result in compaction of the cubes and the lid will close properly. In other words, considering the number of cubes that can be inside the bowl as an evaluation function, shaking the bowl results in a near-maximal solution. The degree of shaking is a parameter of this optimization process and corresponds to the heating and cooling process as described below.

Simulated annealing consists of downhill iteration steps combined with occasional uphill steps that make it possible to escape from local minima. The physical model of the process starts with heating matter until it melts; then the resulting liquid is cooled down slowly to keep a quasi-equilibrium. The cooling algorithm [Metropolis et al., 1953] consists of repeated random displacements (state changes) of atoms in the matter, and the energy change ΔE is evaluated after each state change. If $\Delta E \leq 0$ (lower energy), the state change is accepted, and the new state is used as the starting state of the next step. If $\Delta E > 0$, the state is accepted with probability

$$P(\Delta E) = \exp\left(\frac{-\Delta E}{k_B T}\right). \quad (9.75)$$

To apply this physical model to an optimization problem, the temperature parameter T must be decreased in a controlled manner during optimization. It will be clear that at high values of T , changes that increase energy will be probable, but they become less likely as T declines.

Algorithm 9.13: Simulated annealing optimization

1. Let $\mathbf{x}$ be a vector of optimization parameters; compute the value of the objective function $J(\mathbf{x})$. Initialize T .
2. Perturb $\mathbf{x}$ slightly, creating $\mathbf{x}_{\text{new}}$, and compute the new value of the optimization function $J(\mathbf{x}_{\text{new}})$.
3. Generate a random number $r \in (0, 1)$. If

$$r < \exp\left(\frac{-[J(\mathbf{x}_{\text{new}}) - J(\mathbf{x})]}{k_B T}\right) \quad (9.76)$$

then assign $\mathbf{x} = \mathbf{x}_{\text{new}}$ and $J(\mathbf{x}) = J(\mathbf{x}_{\text{new}})$.

4. Repeat steps 2 and 3 some number of times determined by the cooling schedule.
5. Decrease T according to the cooling schedule, and go to 2.
6. When the cooling schedule is finished, $\mathbf{x}$ represents the solution of the optimization problem.

The sequence of temperatures and number of iterations n_T at each temperature is called the **cooling (or annealing) schedule**. Large values of n_T and small decrements of T yield low final values of the optimization function (the solution is close to the global minimum) but require long computation time. (Remember the sugar bowl example: the shaking is much stronger at the beginning and gradually decreases for the best results). Smaller values of n_T and large decrements in T proceed faster, but the results may not be close to the global minimum. The schedule must be chosen to give a solution close to the minimum without wasting too much computation time, but there is no known practically applicable way to design an optimal annealing schedule, although some general guidelines exist and appropriate parameters can be found for particular problems.

The annealing algorithm is easy to implement, and has been applied to many optimization problems, including pattern recognition, graph partitioning. It has been demonstrated to be of great value, although examples of problems do exist in which it performs less well than standard algorithms and other heuristics. In the computer vision area, applications include stereo correspondence [Barnard, 1987], boundary detection [Geman et al., 1990], texture segmentation [Bouman and Liu, 1991], and edge detection [Tan et al., 1992]. Implementation details and annealing algorithm properties together with an extensive list of references can be found in [van Laarhoven and Aarts, 1987; Otten and van Ginneken, 1989].

9.7 Fuzzy systems

Fuzzy systems are capable of representing diverse, non-exact, uncertain, and inaccurate knowledge or information. They use qualifiers that are very close to the human way of expressing knowledge, such as bright, medium dark, dark, etc. They can represent complex knowledge, and knowledge from contradictory sources. They are based on fuzzy logic, which represents a powerful approach to decision making [Zadeh, 1965; Zimmermann et al., 1984; Kosko, 1992; Cox, 1994; Haykin, 1998; Kecman, 2001].

9.7.1 Fuzzy sets and fuzzy membership functions

When humans describe objects, they often use imprecise descriptors such as *bright*, *large*, *rounded*, *elongated*, etc. For instance, fair-weather clouds may be described as small, medium dark or bright, somewhat rounded regions; thunderstorm clouds may be described as dark or very dark, large regions—people are quite comfortable with such descriptions. However, if the task is to recognize clouds from photographs of the sky automatically by using pattern recognition approaches, it becomes obvious that *crisp* boundaries (discrimination functions) must be drawn that separate the cloud classes. Decisions about boundaries may be quite arbitrary—a decision that a region R_1 characterized by average gray-level g , roundness r and size s represents a thunderstorm cloud, while R_2 characterized by average gray-level $g + 1$, the same roundness r , and size s does not. It may be more appropriate to consider a region R_1 as belonging to the set of fair-weather clouds with some *degree of membership* and belonging to the set of thunderstorm clouds with another degree of membership. Similarly, another region R_2 might belong to both cloud sets with some other degrees of membership. Fuzzy logic facilitates simultaneous membership of regions in different fuzzy sets. Figures 9.26a,b demonstrate the difference between the crisp and fuzzy sets representing the average gray-level of the cloud regions.

A **fuzzy set** S in a fuzzy space X is a set of ordered pairs

$$S = \left\{ (x, \mu_S(x)) | x \in X \right\}, \quad (9.77)$$

where $\mu_S(x)$ represents the grade of membership of x in S . The range of the membership function is a subset of non-negative real numbers whose supremum is finite. For convenience, a supremum of 1 is often used, $\sup_{x \in X} \mu_S(x) = 1$. Fuzzy sets are often denoted solely by the membership function.

The description of DARK regions presented in Figure 9.26b is a classic example of a fuzzy set and illustrates the properties of fuzzy spaces. The **domain** of the fuzzy set is depicted along the x axis and ranges from black to white (0–255), and the degree of membership $\mu(x)$ is along the vertical axis. The membership is between zero (no membership) and one (complete membership). Thus, a white region with an average gray-level of 255 has zero membership in the DARK fuzzy set, while the black region (average gray-level = 0) has complete membership. As shown in Figure 9.26b, the membership function may be linear, but a variety of other curves may also be used (Figure 9.26c,d).

Consider average gray-levels of fair-weather and thunderstorm clouds; Figure 9.27 shows possible membership functions associated with the fuzzy sets DARK, MEDIUM DARK, BRIGHT. As the figure shows, a region with a specific average gray-level g may belong to several fuzzy sets simultaneously. Thus, the memberships $\mu_{\text{DARK}}(g)$,

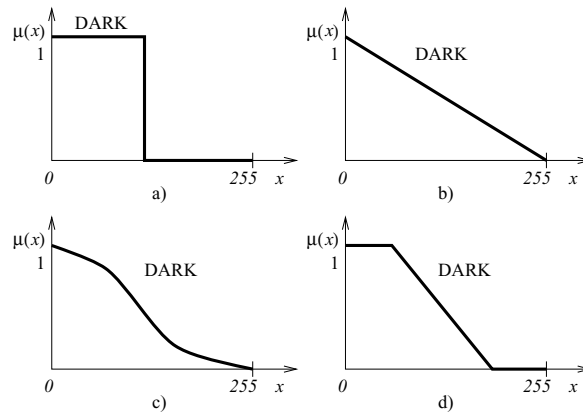


Figure 9.26: Crisp and fuzzy sets representing cloud regions of the same size and roundness, varying average gray-level g . (a) Crisp set showing the Boolean nature of the DARK set. (b) Fuzzy set DARK. (c), (d) Other possible membership functions associated with the fuzzy set DARK. © Cengage Learning 2015.

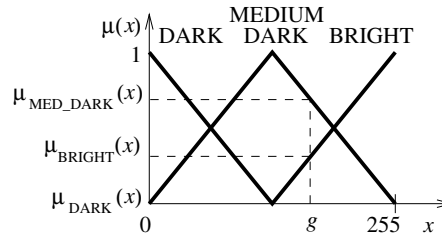


Figure 9.27: Membership functions associated with fuzzy sets DARK, MEDIUM DARK, and BRIGHT. Note that several membership values may be associated with a specific average gray-level g . © Cengage Learning 2015.

$\mu_{\text{MEDIUM DARK}}(g)$, $\mu_{\text{BRIGHT}}(g)$ represent the fuzziness of the description since they assess the degree of certainty about the membership of the region in the particular fuzzy set. The maximum membership value associated with any fuzzy set is called the **height** of the fuzzy set.

In fuzzy system design, normalized versions of membership functions are used. The **minimum normal form** requires at least one element of the fuzzy set domain to have a membership value of one, and the **maximum normal form** is such minimum normal forms for which at least one element of the domain has a membership value of zero.

In fuzzy reasoning systems, membership functions are usually generated in the minimum normal form; there is a wide range of possible fuzzy membership functions, such as linear, sigmoid, beta curve, triangular curve, trapezoidal curve, shouldered curve, arbitrary curve ... [Cox, 1994].

Shapes of fuzzy membership functions can be modified using **fuzzy set hedges**, which can intensify, dilute, form a complement, narrowly or broadly approximate the membership of the fuzzy set elements. Zero or more hedges and the associated fuzzy

set constitute a single semantic entity called a **linguistic variable**. Suppose $\mu_{\text{DARK}}(x)$ represents the membership function of the fuzzy set DARK; then the intensified fuzzy set VERY DARK may have the membership function (Figure 9.28a)

$$\mu_{\text{VERY DARK}}(x) = \mu_{\text{DARK}}^2(x). \quad (9.78)$$

Similarly, a diluting hedge creating a fuzzy set SOMEWHAT DARK may have a membership function (Figure 9.28b)

$$\mu_{\text{SOMEWHAT DARK}}(x) = \sqrt{\mu_{\text{DARK}}(x)}. \quad (9.79)$$

Multiple hedges can be applied to a single fuzzy membership function and a fuzzy set VERY VERY DARK can be created as

$$\mu_{\text{VERY VERY DARK}}(x) = \mu_{\text{DARK}}^2(x) \cdot \mu_{\text{DARK}}^2(x) = \mu_{\text{DARK}}^4(x). \quad (9.80)$$

There are no underlying theoretically solid reasons for these hedge formulae, but they have the merit of success in practice—they simply ‘seem to work’ [Cox, 1994].

9.7.2 Fuzzy set operators

It is unusual that a recognition problem be solved using a single fuzzy set and an associated single membership function. Therefore, tools must be created to combine fuzzy sets, and to determine membership functions of such combinations. In conventional logic, membership functions are either zero or one (Figure 9.26) and for any class set S , a rule of non-contradiction holds: an intersection of a set S with its complement S^c is an empty set

$$S \cap S^c = \emptyset. \quad (9.81)$$

Clearly, this rule does not hold in fuzzy logic, since domain elements may belong to fuzzy sets and their complements simultaneously. There are three basic **Zadeh operators** on fuzzy sets: **fuzzy intersection**, **fuzzy union**, and **fuzzy complement**. Let $\mu_A(x)$ and $\mu_B(y)$ be two membership functions associated with two fuzzy sets A and B with domains X and Y . Then the intersection, union, and complement are pointwise defined for all $x \in X, y \in Y$ (note that other definitions also exist) as

$$\begin{aligned} \text{Intersection } A \cap B : \mu_{A \cap B}(x, y) &= \min(\mu_A(x), \mu_B(y)), \\ \text{Union } A \cup B : \mu_{A \cup B}(x, y) &= \max(\mu_A(x), \mu_B(y)), \\ \text{Complement } A^c : \mu_{A^c}(x) &= 1 - \mu_A(x). \end{aligned} \quad (9.82)$$

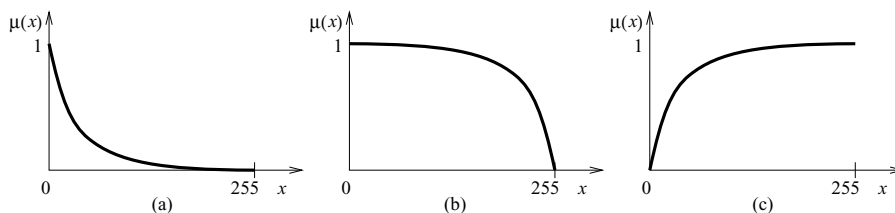


Figure 9.28: Fuzzy set hedges. Fuzzy set DARK is shown in Figure 9.26b. (a) Fuzzy set VERY DARK. (b) Fuzzy set SOMEWHAT DARK. (c) Fuzzy set NOT VERY DARK. © Cengage Learning 2015.

Note that fuzzy set operators may be combined with hedges and new fuzzy sets may be constructed; e.g., a fuzzy set NOT VERY DARK (see Figure 9.28) could be constructed as NOT (VERY (DARK))

$$\mu_{\text{NOT VERY DARK}}(x) = 1 - \mu_{\text{DARK}}^2(x).$$

9.7.3 Fuzzy reasoning

In fuzzy reasoning, information carried in individual fuzzy sets is combined to make a decision. The functional relationship determining the degree of membership in related fuzzy membership functions is called the **method of composition** and results in the definition of a **fuzzy solution space**. To arrive at the decision, a **de-fuzzification** (decomposition) process determines a functional relationship between the fuzzy solution space and the decision. Processes of composition and de-fuzzification form the basis of fuzzy reasoning (Figure 9.29), which is performed in the context of a **fuzzy system model** that consists of control, solution, and working data variables; fuzzy sets; hedges; fuzzy rules; and a control mechanism. Fuzzy models use a series of propositions called fuzzy rules. Unconditional fuzzy rules are of the form

$$x \text{ is } A \quad (9.83)$$

and conditional fuzzy rules have the form

$$\text{if } x \text{ is } A \text{ then } w \text{ is } B, \quad (9.84)$$

where A and B are linguistic variables and x and w represent scalars from their respective domains. The degree of membership associated with an unconditional fuzzy rule is simply $\mu_A(x)$. Unconditional fuzzy propositions are used either to restrict the solution space or to define a default solution space. Since these rules are unconditional, they are applied directly to the solution space by applying fuzzy set operators.

Considering conditional fuzzy rules, there are several approaches to arrive at the decision. **Monotonic fuzzy reasoning** is the simplest approach that can produce a solution directly without composition and de-fuzzification. Suppose x represent a scalar gray-level describing darkness of a cloud, and w the severity of a thunderstorm. The knowledge of thunderstorm severity may be given by the fuzzy rule

$$\text{if } x \text{ is DARK then } w \text{ is SEVERE}. \quad (9.85)$$

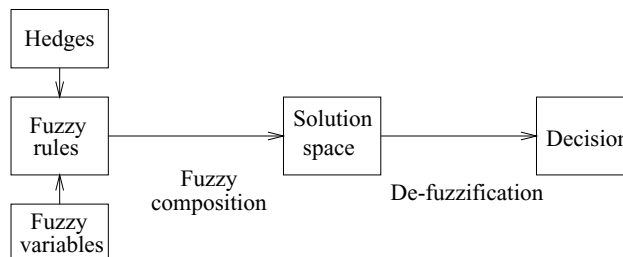


Figure 9.29: Fuzzy reasoning—composition and de-fuzzification. © Cengage Learning 2015.

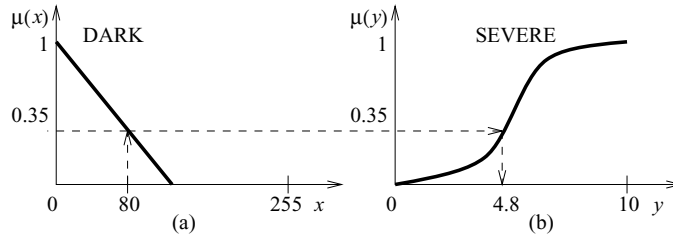


Figure 9.30: Monotonic fuzzy reasoning based on a single fuzzy rule: If the gray-level of the cloud is DARK, then the thunderstorm will be SEVERE. © Cengage Learning 2015.

The algorithm for monotonic fuzzy reasoning is shown in Figure 9.30. Based on determination of the cloud gray-level ($x=80$ in our case), the membership value $\mu_{DARK}(80) = 0.35$ is determined. This value is used to represent the membership value $\mu_{SEVERE}(w) = \mu_{DARK}(x)$ and the decision is made about the expected severity of the thunderstorm; in our case severity $w = 4.8$ on a scale between 0 and 10. This approach may also be applied to complex predicates of the form

$$\text{if } (x \text{ is } A) \bullet (y \text{ is } B) \bullet \dots \bullet (u \text{ is } F) \text{ then } w \text{ is } Z, \quad (9.86)$$

where $\bullet$ represents the conjunctive AND or disjunctive OR operations. Fuzzy intersection and union operators can be used to combine the complex predicates; AND corresponds to fuzzy intersection and OR corresponds to fuzzy union. While the monotonic approach shows the fundamental concept of fuzzy reasoning, it can only be used for a monotonic single fuzzy variable controlled by a single fuzzy rule (possibly with a complex predicate). As the complexity of the predicate proposition increases, the validity of the decision tends to decrease.

Fuzzy composition

Knowledge related to the decision-making process is usually contained in more than one fuzzy rule. A large number of fuzzy rules may take part in the decision-making process and all fuzzy rules are fired in parallel during that process. Clearly, not all fuzzy rules contribute equally to the final solution, and rules that have no degree of truth in their premises do not contribute to the outcome at all. Several composition mechanisms facilitate rule combination.

The most frequently used approach is called the **min-max rule**; a sequence of minimizations and maximizations is applied. First, the minimum of the predicate truth (**correlation minimum**) $\mu_{A_i}(x)$ is used to restrict the consequent fuzzy membership function $\mu_{B_i}(w)$. Let the rules be in the form specified in equation (9.84), and let i represent the i^{th} rule. Then, the consequent fuzzy membership functions B_i are updated in a pointwise fashion and the fuzzy membership functions B_i^+ are formed (Figure 9.31).

$$\mu_{B_i^+}(w) = \min(\mu_{B_i}(w), \mu_{A_i}(x)). \quad (9.87)$$

Second, the pointwise maxima of these minimized fuzzy sets form the solution fuzzy membership function

$$\mu_S(w) = \max_i(\mu_{B_i^+}(w)). \quad (9.88)$$

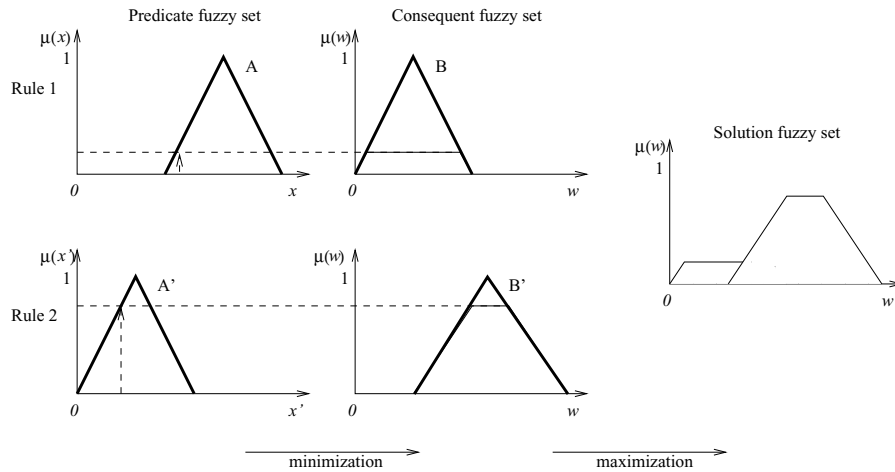


Figure 9.31: Fuzzy min-max composition using correlation minimum. © Cengage Learning 2015.

Figure 9.31 demonstrates the min-max composition process; again, complex predicates may be considered.

The correlation minimum described above is the most common approach to performing the first step of the min-max composition. An alternative approach called **correlation product** scales the original consequent fuzzy membership functions instead of truncating them. While correlation minimum is computationally less demanding and easier to de-fuzzify, correlation product represents in many ways a better method of minimization, since the original shape of the fuzzy set is retained (Figure 9.32).

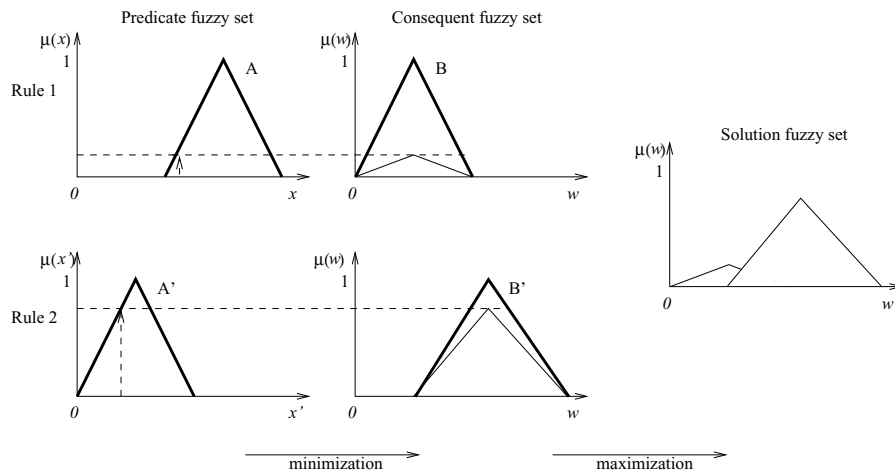


Figure 9.32: Fuzzy min-max composition using correlation product. © Cengage Learning 2015.

De-fuzzification

Fuzzy composition produces a single solution fuzzy membership function for each solution variable. To find the actual crisp solution that will be used for decision making, it is necessary to find a vector of scalar values (one value for each solution variable) that best represents the information contained in the solution fuzzy sets. This process is performed independently for each solution variable and is called de-fuzzification. Two de-fuzzification methods, called **composite moments** and **composite maximum**, are commonly used; many other varieties exist.

Composite moments look for the centroid c of the solution fuzzy membership function—Figure 9.33a shows how the centroid method converts the solution fuzzy membership function into a crisp solution variable c . Composite maximum identifies the domain point with the highest membership value in the solution fuzzy membership function. If this point is ambiguous (on a plateau or if there are two or more equal global maxima), the center of the plateau provides the crisp solution c' (Figure 9.33b). The composite moments approach produces a result that is sensitive to all the rules, while solutions determined using the composite maximum method are sensitive to the membership function produced by the single rule that has the highest predicate truth. While composite moments are used mostly in control applications, recognition applications usually use the composite maximum.

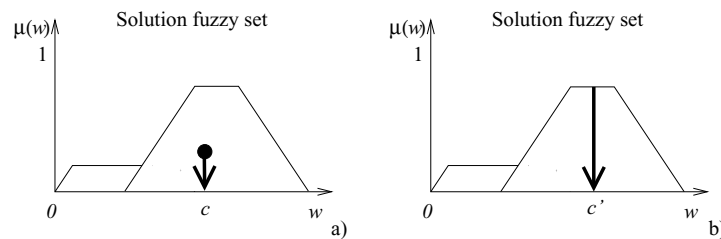


Figure 9.33: De-fuzzification. (a) Composite moments. (b) Composite maximum. © Cengage Learning 2015.

9.7.4 Fuzzy system design and training

Algorithm 9.14: Fuzzy system design

1. Design functional and operational characteristics of the system—determine the system inputs, basic processing approaches, and system outputs. In object recognition, the inputs are patterns and the output represents the decision.
2. Define fuzzy sets by decomposing each input and output variable of the fuzzy system into a set of fuzzy membership functions. The number of fuzzy membership functions associated with each variable depends on the task at hand: typically, an odd number between three and nine is created for each variable. It is recommended that neighboring functions overlap by 10–50%, and that the sum of the membership values of the overlap are less than one.

3. Convert problem-specific knowledge into the fuzzy *if-then* rules that represent a fuzzy associative memory. The number of designed rules is related to the number of input variables. For N variables each of which is divided into M fuzzy membership functions, M^N rules are required to cover all possible input combinations.
4. Perform fuzzy composition and de-fuzzification as described in Section 9.7.3.
5. Using a training set, determine the system's performance. If behavior does not meet the requirements, modify the set descriptions and/or rules and/or the composition and/or de-fuzzification approaches. The speed and success of this fine-tuning step depend on the problem complexity, the designer's level of understanding of the problem, and the level of the designer's experience.

As can be seen from steps 3 and 5 of this algorithm, the design of fuzzy rules may be a tedious and time-consuming process if the rules are to be designed from human experts, as has been typical in most existing applications. Several approaches have been reported that generate fuzzy *if-then* rules automatically using a training set as the source of knowledge and/or for automated adjusting membership functions of fuzzy sets [Ishibuchi et al., 1992, 1995; Abe and Lan, 1995; Homaifar and McCormick, 1995]. Some of the approaches use neural networks or genetic algorithms to control the learning process.

Many applications of fuzzy systems exist in pattern recognition and image processing. In the field of pattern recognition, fuzzy logic has been used for supervised and non-supervised recognition, sequential learning, fuzzy decision theoretic and syntactic classifiers, feature extraction, etc. In image processing and vision, fuzzy logic has been applied to image quality assessment, edge detection, image segmentation, color image segmentation, etc. Extensive work has been done in developing fuzzy geometry approaches [Rosenfeld, 1979, 1985]. Application of fuzzy logic to image segmentation was discussed in Section 7.4. Three-dimensional fuzzy-logic-based analysis of airway trees extracted from X-ray CT image data showed improved performance compared to crisp-rule-based methods [Park et al., 1998]. Use of fuzzy cost functions in optimal graph based image analysis (Section 7.7) was reported in [Haeker et al., 2007; Garvin et al., 2009].

9.8 Boosting in pattern recognition

It is rare that a single classifier will solve a problem completely, or indeed 'well enough'; recognizing this, it is common to combine a number of independent classifiers to improve overall performance. Often, these individual classifiers may be very weak (or **base**) in isolation (that is, in a two-class problem, a classifier may perform little better than 50%). We might train up a number of these simple classifiers, each time working with a different subset of the training examples, and somehow combine this collection of classifiers. This approach to achieving improved performance of any given learning algorithm is called **boosting**. The final combination of weak classifiers in a single classification rule that is much more accurate than any of the constituent rules.

The specifics of this general approach let the weak classifiers function sequentially and place the most weight on 'difficult' training examples, i.e., those that were misclassified

in the previous round(s). To combine the weak rules in a single strong rule, a weighted majority vote of the weak classifier outputs is an obvious strategy. Boosting is then capable of producing very accurate classifications by combining classifications which are only moderately accurate. **AdaBoost** is the best known of these approaches and is widely used [Freund and Schapire, 1997].

Consider a pattern space X , a training set T of n patterns $\mathbf{x}_i$, their corresponding class identifiers ω_i , and assume two-class classification ($\omega_i \in \{-1, 1\}$)

$$T = \{(\mathbf{x}_1, \omega_1), \dots, (\mathbf{x}_n, \omega_n)\}.$$

Suppose we have a supervised learning algorithm W : given a training set T it will deliver a classifier $C_T = W(T)$. The AdaBoost algorithm can then be summarized as Algorithm 9.15 [Schapire, 2002]. Weak classifiers C_k are applied to the training set in which the importance of correctly classifying the individual examples varies step by step, and in each step k is specified by a set of weights $D_k(i)$, so that $\sum_{i=1}^m D_k(i) = 1$. Initially, the weights are uniform but at step $k+1$, the weights of examples incorrectly classified in step k are increased (in a relative sense). Consequently, the weak classifier W_{k+1} concentrates on the difficult examples misclassified in the previous round(s).

Algorithm 9.15: AdaBoost

1. Set $D_1(j) = \frac{1}{n}$, $j = 1, \dots, n$, $i = 1$, and choose K .
2. Sample T_i from T according to D_i , and train $C_i = W(T_i)$.
3. Determine the *error* ϵ_i in C_i . Set $\alpha_i = \frac{1}{2} \log \left(\frac{1-\epsilon_i}{\epsilon_i} \right)$.
4. For all j , set $D_{i+1}(j) = D_i(j)e^{-\alpha_i}$ if C_i classifies $\mathbf{x}_j$ correctly, and $D_{i+1}(j) = D_i(j)e^{\alpha_i}$ otherwise. Renormalize the D_{i+1} .
5. Increment i and go to 2 until $i = K$.
6. Use the classifier

$$C(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^T \alpha_i C_i(\mathbf{x}) \right). \quad (9.89)$$

At step 3, the error is calculated as the probability of the classifier being wrong, *with respect to its training population*:

$$\epsilon_i = \sum_{j: C_i(\mathbf{x}_j) \neq \omega_j} D_i(j).$$

It is assumed that W delivers better than random, so $\epsilon < 0.5$ and hence $\alpha > 0$. α_i serves two purposes:

- It underlies the adjustment to D_i . These increase for ‘difficult’ instances and decrease for ‘easy’ ones.
- It weights the final classifier sum.

Writing $\epsilon_i = \frac{1}{2} - \gamma_i$, then γ measures how much better the classifier is than random (so $\gamma > 0$). It can be shown [Freund and Schapire, 1997] that the final classifier C has an error rate less than

$$\prod_i 2\sqrt{\epsilon_i(1-\epsilon_i)} \leq \exp\left(-2 \sum_i \gamma_i^2\right).$$

Most problems are not binary: patterns may belong to one of a number of classes. More generally, a pattern may belong to *several* of a number of classes. For example, classifying documents as *news*, *politics*, *sport*, *entertainment*, $\dots$, a given item may belong to more than one category. Adaboost can be adapted to these more general problems.

Algorithm 9.16: AdaBoost-MH

1. Take $T = \{(\mathbf{x}_1, A_1), \dots, (\mathbf{x}_n, A_n)\}$, $A_j \subset Y = [1, k]$. Set $D_1(j, l) = \frac{1}{nk}$ for $j = 1, \dots, n$, $l = 1, \dots, k$, and choose K .
2. Sample T_i from T according to D_i , and train $C_i = W(T_i)$. The sign of $C_i(\mathbf{x}, j)$ indicates whether label j attaches to $\mathbf{x}$ or not.
3. Determine α_i .
4. **Update:** set $D_{i+1}(j, l) = D_i(j, l)e^{-\alpha_i C_i(\mathbf{x}_j, l)}$ if C_i classifies $\mathbf{x}_j$ correctly, and $D_{i+1}(j) = D_i(j)e^{\alpha_i C_i(\mathbf{x}_j, l)}$ otherwise.
5. Renormalize the $D_{i+1}(j, l)$.
6. Increment i and go to 2 until $i = K$.
7. Use

$$C(\mathbf{x}, l) = \text{sign} \left(\sum_{i=1}^T \alpha_i C_i(\mathbf{x}, l) \right). \quad (9.90)$$

In addition to an ability to learn from examples, the ability to generalize and thus correctly classify previously unseen patterns is of basic importance. Theoretical considerations suggest that Adaboost may be prone to over-fitting, but experimental results show that typically it does not overfit, even when run for thousands of rounds. More interestingly, it was observed that AdaBoost sometimes continues to drive down the classification error long after the training error had already reached zero (see Figure 9.34). The error decrease can be associated with an increase in the *margin* that was introduced in association with SVMs (Section 9.2.5), which explicitly maximizes the minimum margin. In boosting, the margin is a number from the interval $[-1, 1]$ that is positive only if the strong classifier correctly classifies the pattern. The magnitude of the margin corresponds to the level of confidence in the classification.

Many modifications of AdaBoost exist. Incorporating a priori knowledge in the boosting scheme is introduced in [Rochery et al., 2002]. AdaBoost has the ability to identify *outliers* which are inherently difficult to classify; to deal with a possible decrease in AdaBoost's performance when a large number of outliers exists, *Gentle AdaBoost* and *BrownBoost* were introduced in [Friedman et al., 2000; Freund, 2001], in which the outliers' influence is less pronounced.

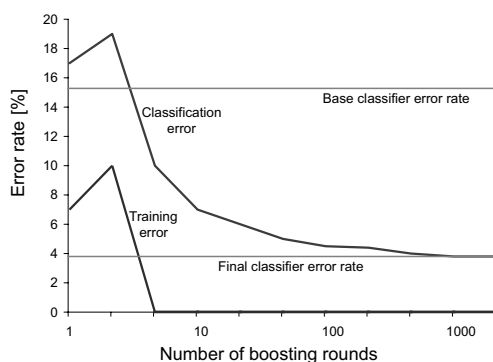


Figure 9.34: AdaBoost: training and testing error rate curves plotted against the number of boosting rounds. Note that testing error keeps decreasing long after training error has already reached zero. This is associated with a continuing increase in the margin that increases the overall classification confidence with additional rounds of boosting. © Cengage Learning 2015.

A very well known application of the boosting approach is the Viola-Jones face detector, explained in detail in Section 10.7. This in turn can be exploited as a pedestrian detector in video, described in Section 16.4.

In AdaBoost, the weak classifiers are frequently statistical in nature, but there is no special requirement on them. For example, rule-based classifiers may be considered. The individual weak classifiers do not even have to be of the same kind. This flexibility further increases the general character of boosting strategies when applied to pattern recognition problems.

9.9 Random forests

Random forests are a classification approach that is especially well suited for problems with many classes when large datasets are available for training [Breiman, 2001]. That they naturally deal with more than two classes, provide probabilistic outputs, offer excellent unseen data generalization, and are inherently parallel are all very intriguing characteristics of this approach [Criminisi and Shotton, 2013].

Here, we give a general overview of the underlying methodology, and a description of their use in image understanding is given in Section 10.8. Their treatment here is relatively brief, but many focused research articles devoted to them are easy to locate, starting with seminal publications [Breiman et al., 1984; Ho and Ohnishi, 1995]. The work of Quinlan offered an approach called the C4.5 algorithm to train decision trees optimally [Quinlan, 1993]. The concept of a single decision tree was extended to multiple such trees in a randomized fashion, forming random forests. Some aspects of random forests resemble the boosting strategy described in Section 9.8 since weak classifiers are associated with individual tree nodes and the entire forest yields a strong classification decision. This presentation is primarily based on [Criminisi et al., 2011], in which discussion of recent developments and additional implementation details can be found.

Random forests are used for two main decision making tasks: *classification* and *regression*. In classification (e.g., when classifying images into categories denoting types of captured scenes – beach, road, person, etc.), the decision-making output is a class label. In non-linear regression (e.g., predicting severity of flu season from – possibly multi-dimensional – social network data), the outcome is a continuous numeric value.

A *decision tree* consists of internal (or split) nodes and terminal (or leaf) nodes (see Figure 9.35). Arriving image patterns are evaluated in respective nodes of the tree and – based on the pattern properties—are passed to either left or right child nodes. Leafs L store the statistics of the patterns that arrived at a particular node during training. When a decision tree $\mathcal{T}_t$ is used for classification, the stored statistical information contains the probability of each class ω_r , $r \in 1, \dots, R$ or $p_t(\omega_r|L)$. If used for regression, the statistical information contains a distribution over the continuous parameter that is being estimated. For a combined *classification–regression* task (Section 10.8), both kinds of statistics are collected. A random forest $\mathcal{T}$ then consists of a set of T such trees and each tree $\mathcal{T}_t$, $t \in \{1, \dots, T\}$, is trained on a randomly sampled subset of the training data. It has been shown that ensembles of slightly different trees (differences resulting, e.g., from training on random training subsets) produce much higher accuracy and better noise insensitivity compared to single trees when applied to previously unseen data, demonstrating excellent generalization capabilities.

Once a decision tree is trained, predefined binary tests are associated with each internal node and unseen data patterns are passed from the tree root to one of the leaf nodes. The exact path is decided based on the outcome of the internal-node tests, each of which determines whether the data pattern is passed to one or the other child node. The process of binary decisions is repeated until the data pattern reaches a leaf node. Each of the leaf nodes contains a *predictor*, i.e., a classifier or a regressor, which associates the pattern with a desired output (classification label, regression value). If a forest of many trees is employed, the individual tree leaf predictors are combined to form a single prediction. In this sense, the decision-making process based on the node-associated binary predictors is fully deterministic.

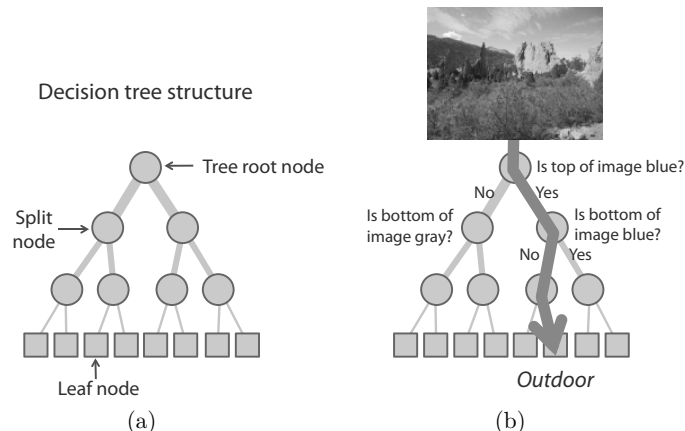


Figure 9.35: Decision tree. (a) Decision trees contain one root node, internal or split nodes (circles), and terminal or leaf nodes (squares). (b) A pattern arrives at the root and is sequentially passed to one of two children of each split node according to the node-based split function until it reaches a leaf node. Each leaf node is associated with a probability of a specific decision, for example associating a pattern with a class label. Based on [Criminisi et al., 2011]. A color version of this figure may be seen in the color inset—Plate 16.

9.9.1 Random forest training

The decision-making capabilities of the individual tree nodes will depend on the predefined binary tests associated with each internal node and on the leaf predictors. The parameters of the binary tests can be either expert-designed or result from training. Let S_i denote the subset of training data reaching node i , and S_i^L and S_i^R denote subsets of training data reaching the left or right child nodes of node i , respectively. Since the decisions at each node are binary, we have:

$$S_i = S_i^L \cup S_i^R, \quad S_i^L \cap S_i^R = \emptyset. \quad (9.91)$$

The training process constructs a decision tree for which parameters of each binary test were chosen to minimize some objective function. To stop construction of tree children at a certain node of a certain branch, tree-growth stopping criteria are applied. If the forest contains T trees, each tree $\mathcal{T}_t$ is trained independently of the others using a randomly selected subset of the training set per tree.

Take a 4-class classification problem, in which the same number of 2D patterns belong to each class (Figure 9.36). Comparing two of many ways in which the feature space may be split—say, using a half-way horizontal or half-way vertical split line—it can be seen

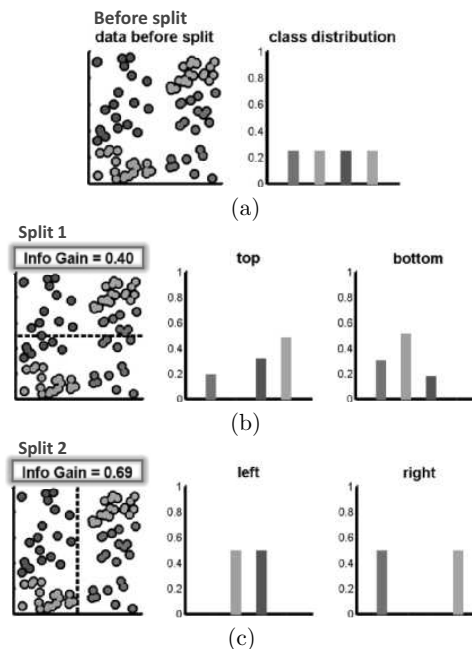


Figure 9.36: Information gain resulting from a split. (a) Class distributions prior to the split. (b) Distributions after a horizontal split. (c) Distributions after a vertical split. Note that both yield more homogeneous subsets and that the entropy of both subsets is decreases as a result of these splits. Based on [Criminisi et al., 2011]. A color version of this figure may be seen in the color inset—Plate 17.

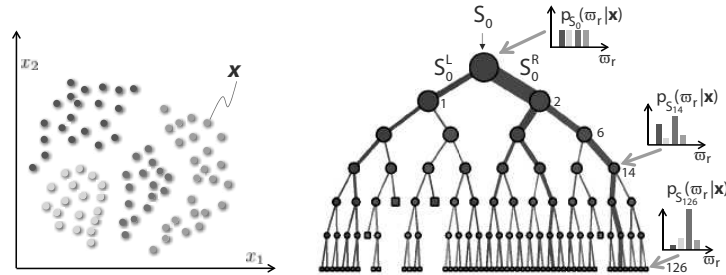


Figure 9.37: Tree training. Distribution of two-dimensional feature patterns in the feature space is reflected by the class distribution at the root-node level. Here class labels are color-coded and each class includes an identical number of patterns. As a result of training, binary decision functions associated with each split node are optimized—note the increased selectivity of class distributions at nodes more distant from the root (reflecting decreasing entropy). Relative numbers of training patterns passing through individual tree branches are depicted by their thickness. The branch colors correspond to the distribution of class labels. *Based on [Criminisi et al., 2011]. A color version of this figure may be seen in the color inset—Plate 18.*

that both yield more homogeneous subsets (higher similarity of subset-member patterns) and result in a lower entropy of the subsets (Section 2.3.3) than was the case prior to the splits. The change in entropy (equation 2.7), called the *information gain* I , is

$$I = H(S) - \sum_{i \in \{1,2\}} \frac{|S^i|}{|S|} H(S^i). \quad (9.92)$$

Clearly the vertical split in Figure 9.36 separates the classes much better than the horizontal split and this observation is reflected in the differences in the information gain. The parameters of the internal-node binary decision elements can be set so that the information gain achieved on the training set by each split is maximized. Forest training is based on this paradigm.

As shown in Figure 9.37, the binary *split function* associated with a node j

$$h(\mathbf{x}, \theta_j) \in \{0, 1\} \quad (9.93)$$

directs the patterns $\mathbf{x}$ arriving at node j to either the left or the right child (0 or 1 decision). These node-associated split functions play the role of weak classifiers (see also Section 9.8). The weak learner at node j is characterized by parameters $\theta_j = (\phi_j, \psi_j, \tau_j)$ defining the feature selection function ϕ (specifying which features from the full feature set are used in the split function associated with node j), data separation function ψ (which hypersurface type is used to split the data, e.g., axis-aligned hyperplane, oblique hyperplane, general surface, etc.—see Figure 9.38), and threshold τ driving the binary decision.

Parameters θ_j must be optimized for all tree nodes j during training, yielding optimized parameters θ_j^* . One way to optimize the split function parameters is to maximize the information gain objective function

$$\theta_j^* = \operatorname{argmax}_{\theta_j} I_j, \quad (9.94)$$

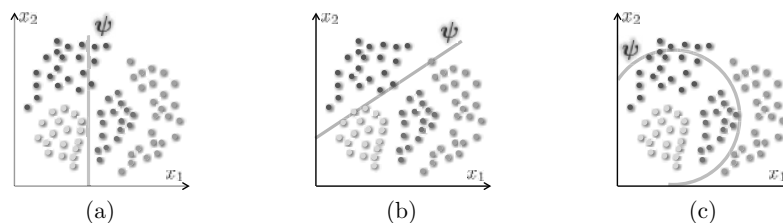


Figure 9.38: Weak learners can use a variety of binary discrimination functions. (a) Axis-aligned hyperplane. (b) General hyperplane. (c) General hypersurface. Based on [Criminisi et al., 2011]. A color version of this figure may be seen in the color inset—Plate 19.

where $I_j = I(S_j, S_j^L, S_j^R, \theta_j)$ and S_j, S_j^L, S_j^R represent training data before and after the left/right split at node j .

The decision tree is constructed during the training and a *stopping criterion* is needed for each tree node to determine whether child-nodes should be formed, or tree-branch construction terminated. Meaningful criteria include: defining a maximum allowed tree depth D (this is very popular); or allowing the node to form child nodes only if a pre-specified minimum information gain is achieved by a split during training; or not allowing child-node construction if a node is not on a frequented data path, i.e., if a node processes less than a pre-defined number of training patterns.

After training, tree leaf nodes contain information that will be used during the decision making process. If the goal is *classification*, each leaf node L stores the empirical distribution over the classes associated with a training subset that has reached that particular leaf node. Considering R classes, a probabilistic predictor model for pattern $\mathbf{x}$ and tree $\mathcal{T}_t$ is

$$p_t(\omega_r|\mathbf{x}), \text{ where } r \in \{1, \dots, R\}. \quad (9.95)$$

If a **regression** task is considered, the output is a continuous variable. The leaf predictor model yields a posterior over the desired continuous variable.

A forest $\mathcal{T}$ consists of T trees that are randomly different from each other. The randomness is introduced during training, for example by employing a random subset of the training data for each tree of the forest (so called *bagging*), using randomized node optimization, or other approaches.

9.9.2 Random forest decision making

The behavior of the random forest decision process depends on the number T of trees forming the forest, the maximum allowed tree depth D , parameters of the randomness, choice of the weak learner model, objective function used for tree training, and of course the choice of features representing the underlying data.

The training described above is performed independently for each tree and may be performed in parallel. When using a forest consisting of T trees, the testing-set pattern $\mathbf{x}$ is simultaneously provided to all T roots of T trained trees and the tree-level processing can also be performed in parallel until the pattern reaches the tree leaves, each of which is providing a leaf-specific prediction. All tree predictions must be combined to form a single forest prediction that represents the forest output. The forest prediction can be obtained in several ways, for example by averaging all tree predictions or multiplying the

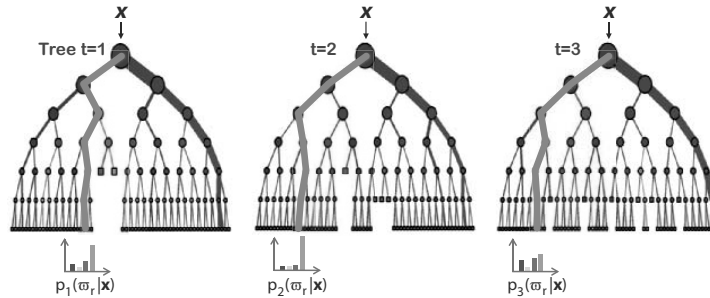


Figure 9.39: Decision making process using classification random forests—the same unseen pattern $\mathbf{x}$ arrives simultaneously at the roots of all trees forming the forest. At each node at which the pattern arrives, a test optimized during training is employed to direct the pattern to one of the two children. Due to the random character of training, each tree of the forest includes different split functions and the same pattern follows a different path through the tree nodes until a leaf is reached. Once the leaf is reached in each tree, the tree-specific class posteriors $p_t(\omega_r|\mathbf{x})$ are averaged (or multiplied) to yield a forest posterior $p(\omega_r|\mathbf{x})$. Note that there is one and only one path how a specific pattern $\mathbf{x}$ passes through the tree during decision making. Consequently, the probability $p_t(\omega_r|\mathbf{x})$, while linked to a single leaf node, represents the posterior associated with the entire tree. Based on [Criminisi et al., 2011]. A color version of this figure may be seen in the color inset—Plate 20.

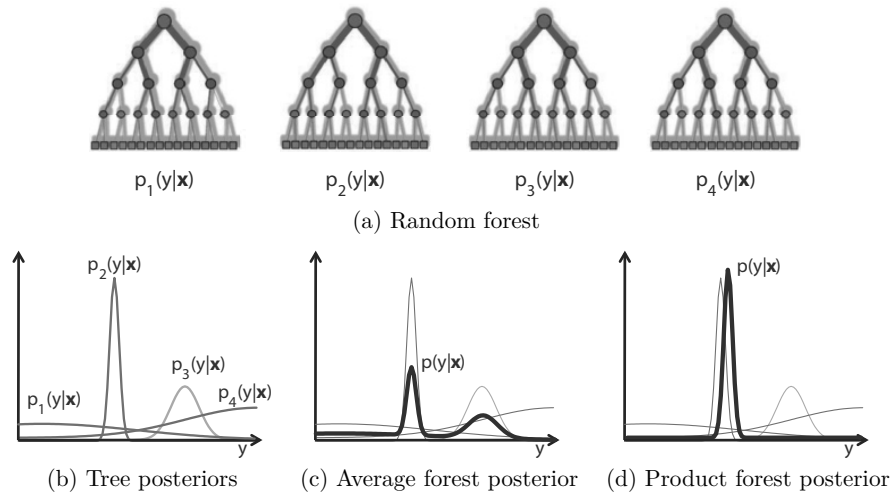


Figure 9.40: Random forest ensemble model for predicting continuous variable y . (a) Random forest consisting of 4 trees. (b) Posteriors of individual trees $p_t(y|\mathbf{x})$. (c) Forest posterior resulting from average-based ensemble model. (d) Forest posterior resulting from multiplication-based ensemble model. Note the strongest influence of the more informative trees on the forest posterior. Based on [Criminisi et al., 2011]. A color version of this figure may be seen in the color inset—Plate 21.

tree outputs together:

$$p(\omega_r|\mathbf{x}) = \frac{1}{T} \sum_{t=1}^T p_t(\omega_r|\mathbf{x}) \quad \text{or} \quad p(\omega_r|\mathbf{x}) = \frac{1}{Z} \prod_{t=1}^T p_t(\omega_r|\mathbf{x}), \quad (9.96)$$

where $1/Z$ provides probabilistic normalization.

Tree **ensemble models** combine contributions from many trees into a single forest-level output. Figure 9.39 depicts outputs from three randomly trained decision trees used for a classification task, these outputs are combined according to equation (9.96). Figure 9.40 shows how prediction outputs from multiple trees ($T = 4$) can be combined for a regression-task random forest. Let the posterior outputs representing prediction of the desired continuous variable y for all four individual trees for pattern $\mathbf{x}$ be $p_t(y|\mathbf{x})$, $t \in \{1, \dots, 4\}$. Figure 9.40c shows the forest output obtained via averaging of individual tree outputs, and Figure 9.40d gives the output obtained by multiplying tree outputs. In both panels, the forest output benefits from combining the outputs and is influenced more strongly by the more confident portions of the individual tree outputs. Individual tree contributions may be noisy and averaging many tree posteriors decreases noise sensitivity. Understandably, product-based ensemble models are more sensitive to noise.

9.9.3 Random forest extensions

Random forest approaches have been shown highly applicable to supervised, unsupervised, and semi-supervised learning as well as for manifold learning using manifold forests. Discussion of such extensions and description of the effect of random forest parameters on their performance, accuracy, and generalization can be found in [Criminisi et al., 2011; Criminisi and Shotton, 2013]. The same publications also contain useful comparisons with support vector machines (Section 9.2.5), boosting approaches (Section 9.8), RANSAC (Section 10.3), and Gaussian mixture models/expectation-maximization (Section 10.13). Section 10.8 gives several practical examples demonstrating how random forests can be used for image analysis and understanding tasks.

9.10 Summary

- **Object recognition, pattern recognition**
 - Pattern recognition is used for region and object **classification**, and represents an important building block of complex machine vision processes.
 - No recognition is possible without **knowledge**. Knowledge about objects being processed, and hierarchically higher and more general knowledge about object classes is required.
- **Knowledge representation**
 - Descriptions and features
 - Grammars and languages
 - Predicate logic

- Production rules
- Fuzzy logic
- Semantic nets
- Frames, scripts

- **Statistical pattern recognition**

- **Object recognition** is based on assigning classes to objects, using a **classifier**. The number of classes is usually known beforehand, and typically can be derived from the problem specification.
- The classifier will use **patterns** sensed from the object to take decisions.
- Nearest neighbor classifiers are straightforward to understand and widely used. They are expensive to use for high-dimensional or large datasets, and have been improved via K-D trees and approximation approaches.
- For statistical pattern recognition, a **quantitative** description of objects using elementary numerical descriptions—**features**—is used. The set of possible patterns forms a **pattern space** or **feature space**. The **classes** form clusters in this space, which can be separated by **discrimination hyper-surfaces**.
- A **statistical classifier** is a device with n inputs and 1 output. Each input is given by one of n features measured from an object to be classified. An R -class classifier generates one of R symbols ω_r , the **class identifiers**.
- Classification parameters are determined from a **training set** of examples during **classifier learning**. Two common learning strategies are **probability density estimation** and **direct loss minimization**.
- **Support vector machine** training is based on maximization of **margin** between two classes; non-linear classification is facilitated by the **kernel trick**. Multi-class classification is achieved by combining multiple 2-class classifiers.
- **Cluster analysis** is one of many methods that do not need training sets for learning. It divides the set of patterns into subsets (clusters) based on mutual similarity of subset elements.

- **Neural nets**

- Most neural approaches are based on combinations of elementary processors (**neurons**), each of which takes a number of inputs and generates a single output. Each input has a weight, and output is a function of the weighted sum of inputs.
- **Feed-forward** networks are common in pattern recognition problems. Training uses a set of examples and is often based on **back-propagation**.
- **Self-organizing** networks do not require a training set to cluster the patterns.
- **Hopfield** networks do not have designated inputs and outputs; rather, the current configuration represents the state. The network acts as an associative memory where *exemplars* are stored.

- **Syntactic pattern recognition**

- Syntactic pattern recognition uses a **qualitative** description of objects. Elementary properties of objects are called **primitives**. **Relational structures** are used to describe relations between the object primitives.
- The set of all primitives is called the **alphabet**. The set of all words in the alphabet that can describe objects from one class is called the **description language**. A **grammar** is a set of rules that must be followed when words of the specific language are constructed from the alphabet.
- Grammar construction usually requires significant human interaction. In simple cases, an automated process called **grammar inference** can be applied.
- The recognition decision of whether or not the word can be generated by a particular grammar is made during **syntactic analysis**.

- **Recognition as graph matching**

- Matching of a model and an object graph description can be used for recognition. An exact match of graphs is called graph **isomorphism**. Determination of graph isomorphism is computationally expensive.
- In the real world, the object graph usually does not match the model graph exactly—**isomorphisms** cannot assess the level of mismatch. **Graph similarity** may identify objects represented by similar graphs.

- **Optimization techniques in recognition**

- Optimization problems seek minimization or maximization of an **objective function**. Design of the objective function is a key factor in performance.
- Most conventional approaches to optimization use calculus-based **hill climbing** methods. Search can easily end in local rather than global extrema.
- **Genetic algorithms** use natural evolution mechanisms (*‘survival of the fittest’*) to search for the maximum. Potential solutions are represented as strings. Genetic algorithms search from a population of potential solutions, not a single solution. The sequence of **reproduction**, **crossover**, and **mutation** generates a new population of strings from the previous population. The fittest string represents the final solution.
- **Simulated annealing** combines two basic optimization principles, **divide and conquer** and **iterative improvement** (hill climbing). This combination avoids getting stuck in local optima.

- **Fuzzy systems**

- Fuzzy systems are capable of representing diverse, non-exact, uncertain, and inaccurate knowledge or information. They use qualifiers that are very close to the human way of expressing knowledge.
- Fuzzy reasoning is performed in the context of a **fuzzy system model** that consists of control, solution, and working data variables; fuzzy sets; hedges; fuzzy rules; and a control mechanism.

- **Fuzzy sets** represent properties of fuzzy spaces. **Membership functions** represent the fuzziness of the description and assess the degree of certainty about the membership of an element in the particular fuzzy set. Shape of fuzzy membership functions can be modified using **fuzzy set hedges**. A hedge and its fuzzy set constitute a single semantic entity called a **linguistic variable**.
 - **Fuzzy if-then rules** represent fuzzy associative memory in which knowledge is stored.
 - In fuzzy reasoning, information carried in individual fuzzy sets is combined to make a decision. The functional relationship determining the degree of membership in related fuzzy regions is called the **method of composition** and results in definition of a **fuzzy solution space**. To arrive at the decision, **de-fuzzification** is performed. Processes of composition and de-fuzzification form the basis of fuzzy reasoning.
- **Boosting**
 - Boosting achieves an improved classification performance by combining classification outcomes of a (large) number of ‘weak’ classifiers of moderate performance.
 - A single complex classification rule is replaced with a number of simple rules, which may perform only slightly better than random selection. Boosting is capable of producing very accurate classifications by combining outcomes of only moderately accurate classifiers.
 - AdaBoost is a widely used boosting algorithm, in which weak classifiers are sequentially trained on the training set, with each next classifier being trained on a differently weighted set of training patterns. Weighting is set according to the difficulty of classifying individual patterns, assessed according to the classification outcomes in previous steps.
 - The outcomes of all weak classifiers are combined to form a final strong classifier. The combination is based on a weighted majority vote.
 - There are no other requirements on the weak classifier selection than that they need to provide a better than random classification performance.
 - **Random Forests**
 - Random forests are especially well suited for **problems with many classes** when **large datasets** are available for training.
 - Random forests are used for two main decision making tasks: **classification** and **regression**.
 - In **classification**, the decision-making output is a class label.
 - In **non-linear regression**, the outcome is a continuous numeric value.
 - Each tree of a forest can be trained in parallel. Once trained, predefined binary tests are associated with each internal node and previously unseen data patterns are passed from the tree root to a leaf nodes based on the outcome of the internal-node tests.

- A **random forest** consists of a set of decision trees, each of which may be trained on a randomly sampled subset of the training data.
- **Ensembles** of slightly different trees produce much higher accuracy and better noise insensitivity compared to single trees when employed for decision making.

9.11 Exercises

Short-answer questions

- S9.1 Define the *syntax* and *semantics* of knowledge representation.
- S9.2 Describe the following knowledge representations, giving for each one at least one example that is different from examples given in the text: *Descriptions* (features), *Grammars*, *Predicate logic*, *Production rules*, *Fuzzy logic*, *Semantic nets*, *Frames* (*scripts*).
- S9.3 Define the following terms: *Pattern*, *Class*, *Classifier*, *Feature space*.
- S9.4 Define the terms: *Class identifier*, *Decision rule*, *Discrimination function*.
- S9.5 Explain the main concepts and derive a mathematical representation of the discrimination functions for:
- A minimum distance classifier
 - A minimum error classifier
- S9.6 What is a training set? How is it designed? What influences its desired size?
- S9.7 Explain the principle of a Support Vector Machine (SVM) classifier.
- S9.8 Explain why learning should be inductive and sequential.
- S9.9 Describe the conceptual differences between supervised and unsupervised learning.
- S9.10 Draw schematic diagrams of a feed-forward and Hopfield neural networks. Discuss their major architectural differences.
- S9.11 For what is the back-propagation algorithm used? Explain its main steps.
- S9.12 What is the reason for including the momentum constant in back-propagation learning?
- S9.13 Explain the functionality of Kohonen neural networks. How can they be used for unsupervised pattern recognition?
- S9.14 Explain how Hopfield networks can be used for pattern recognition.
- S9.15 Define the following terms: *Primitive*, *Alphabet*, *Description language*, *Grammar*.
- S9.16 Describe the main steps of syntactic pattern recognition.
- S9.17 Give a formal definition of a grammar.
- S9.18 When are two grammars equivalent?
- S9.19 What is grammar inference? Give its block diagram.
- S9.20 Formally define: *A graph*, *Graph isomorphism*, *Subgraph isomorphism*, *Double subgraph isomorphism*.
- S9.21 Define Levenshtein distance. Explain its application to assessing string similarity.
- S9.22 Explain why hill-climbing optimization approaches may converge to local instead of global optima.

- S9.23 Explain the concept and functionality of genetic algorithm optimization. What are the roles of reproduction, crossover, and mutation in genetic algorithms?
- S9.24 Explain the concept of optimization based on simulated annealing. What is the annealing schedule?
- S9.25 List the advantages and disadvantages of genetic algorithms and simulated annealing compared to optimization approaches based on derivatives.
- S9.26 Define the terms:
- (a) Fuzzy set
 - (b) Fuzzy membership function
 - (c) Minimum normal form of a fuzzy membership function
 - (d) Maximum normal form of a fuzzy membership function
 - (e) Fuzzy system
 - (f) Domain of a fuzzy set
 - (g) Hedge
 - (h) Linguistic variable
- S9.27 Use Zadeh's definitions to define formally: *Fuzzy intersection*, *Fuzzy union*, *Fuzzy complement*.
- S9.28 Explain fuzzy reasoning based on composition and de-fuzzification. Draw a block diagram of fuzzy reasoning.
- S9.29 Explain the rationale behind using weak classifiers in the Adaboost training and classification process.
- S9.30 The training process in boosting is sequential. Why is increasing weight placed on the training examples that were misclassified in the previous steps?
- S9.31 Provide a flowchart outlining the training process for a single tree of depth $D = 3$ from a random forest. Using the flowchart, identify a path that a hypothetical single training pattern follows and describe how this training pattern contributes to the training process.
- S9.32 Give an example of deriving a random forest posterior from posteriors of at least 3 individual trees:
- (a) Using average-based ensemble model.
 - (b) Using multiplication-based ensemble model.

Problems

Create the following training and testing sets of feature vectors to use in some of the Problems that follow.

TRAIN1:

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	2	4	3	3	4	10	9	8	9	10
x_2	3	2	3	2	3	7	6	6	7	6

TEST1:

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	3	6	5	5	6	13	12	11	11	13
x_2	5	3	4	3	5	10	8	8	9	8

TRAIN2:

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	2	6	-2	7	5	-2	-6	2	-4	-5
x_2	400	360	520	-80	180	-200	-200	-400	-600	-400

ω_i	ω_3	ω_3	ω_3	ω_3	ω_3
x_1	-10	-8	-15	-12	-14
x_2	200	140	100	50	300

TEST2:

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	4	8	-3	9	6	-1	-4	3	-2	-3
x_2	600	540	780	-120	270	-250	-250	-470	-690	-470

ω_i	ω_3	ω_3	ω_3	ω_3	ω_3
x_1	-15	-13	-6	-17	-16
x_2	230	170	130	80	450

P9.1 Let a minimum distance classifier be used to recognize two-dimensional patterns from three classes K_1, K_2, K_3 . The training set consists of five patterns from each class:

$$K_1 := \left\{ \begin{pmatrix} 0 \\ 6 \end{pmatrix}, \begin{pmatrix} 1 \\ 6 \end{pmatrix}, \begin{pmatrix} 2 \\ 6 \end{pmatrix}, \begin{pmatrix} 1 \\ 5 \end{pmatrix}, \begin{pmatrix} 1 \\ 7 \end{pmatrix} \right\},$$

$$K_2 := \left\{ \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \end{pmatrix} \right\},$$

$$K_3 := \left\{ \begin{pmatrix} 8 \\ 6 \end{pmatrix}, \begin{pmatrix} 9 \\ 6 \end{pmatrix}, \begin{pmatrix} 10 \\ 6 \end{pmatrix}, \begin{pmatrix} 9 \\ 5 \end{pmatrix}, \begin{pmatrix} 9 \\ 7 \end{pmatrix} \right\}.$$

Determine (sketch) the discrimination functions in the two-dimensional feature space.

P9.2 Use some software to generate data from two or more Gaussian distributions: these could be 1D but it is more interesting to use 2D or higher dimension. Use these data to define training and test sets for 'leave one out' testing of an NN or kNN classifier. Summarize the results for various selections of Gaussians and values of k .

P9.3 Let a minimum error classifier be used to recognize two-dimensional patterns from two classes, each having a normal distribution $N(\boldsymbol{\mu}_r, \boldsymbol{\Psi}_r)$:

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}, \quad \boldsymbol{\Psi}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{\mu}_2 = \begin{pmatrix} 4 \\ 3 \end{pmatrix}, \quad \boldsymbol{\Psi}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Assume unit loss functions and equal a priori probabilities of classes $P(\omega_1) = P(\omega_2) = 0.5$. Determine (sketch) the discrimination function in the two-dimensional feature space.

P9.4 Repeat Problem P9.3 considering modified parameters of the normal distributions:

$$\mu_1 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}, \quad \Psi_1 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \quad \mu_2 = \begin{pmatrix} 4 \\ 3 \end{pmatrix}, \quad \Psi_2 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}.$$

P9.5 Repeat Problem P9.4 with $P(\omega_1) = P$, $P(\omega_2) = (1 - P)$. Show how the discrimination function locations in the two-dimensional feature space change as a function of P .

P9.6 Consider the training set specified in Problem P9.1. Assume that the three pattern classes have normal distributions and that a priori probabilities of classes are equal $P(\omega_1) = P(\omega_2) = P(\omega_3) = 1/3$. Determine (sketch) the discrimination functions of the minimum error classifier in the two-dimensional feature space. Discuss under what circumstances the discrimination functions of a minimum distance classifier are identical to discrimination functions of the minimum error classifier if both were trained using the same training set.

P9.7 Develop a program for training and classification using the minimum distance classifier. Assess classification correctness.

- (a) Train and test using data sets TRAIN1 and TEST1.
- (b) Train and test using data sets TRAIN2 and TEST2.

P9.8 Develop a program for training and classification using the minimum error classifier, considering unit loss functions. Assume the training data have normal distribution. Assess classification correctness.

- (a) Train and test using data sets TRAIN1 and TEST1.
- (b) Train and test using data sets TRAIN2 and TEST2.

P9.9 Develop a program for cluster analysis using the k -means approach. Vary the initialization of cluster starting points and explore its influence on clustering results. Vary the number of classes and discuss the results.

- (a) Use a combined data set TRAIN1 and TEST1.
- (b) Use a combined data set TRAIN2 and TEST2.

P9.10 Create a training set and a testing set of feature vectors using the shape description program developed in Problem P8.13 to determine shape feature vectors. Use simple shapes (e.g., triangles, squares, rectangles, circles, etc.) of different sizes. Select up to five discriminative features to form the feature vectors of analyzed shapes. The training as well as the testing sets should consist of at least 10 patterns from each class. The training and testing sets will be used in the experiments below.

P9.11 Apply the program developed in Problem P9.7 to the training and testing sets created in Problem P9.10. Assess classification correctness.

P9.12 Apply the program developed in Problem P9.8 to the training and testing sets created in Problem P9.10. Assume normal distributions, and that you have a sufficient number to determine representative dispersion matrices and mean values from the training set. Assess classification correctness in the testing set. Compare with the performance of the minimum distance classifier used in Problem P9.11.

P9.13 Apply the program developed in Problem P9.9 to the testing set created in Problem P9.10. First, assume that the number of classes is known. Assess clustering correctness and compare it with that of the supervised methods used in Problems P9.11 and P9.12. Then, vary the initialization of cluster starting points and explore the influence on clustering results.

- P9.14 Download a program that performs back-propagation learning and classification (there are many to be found). Train and test a three-layer feed-forward neural network using artificial data from a two-dimensional feature space representing patterns from at least three separable classes.
- P9.15 Apply the program of Problem P9.14 to the testing set created in Problem P9.10. Assess classification correctness and compare it with that of the statistical classification methods assessed in Problems P9.11 and P9.12.
- P9.16 Choice of width of layers is often a problem. Repeat Problems P9.14 and P9.15, paying attention to the size of the hidden layer. Draw some conclusions about network performance and training time as the size of this layer varies.
- P9.17 Implement Algorithm 9.7. For some datasets devised by you, or extracted from some known application, run the algorithm. Compare its performance for different sizes and topologies of output layer, and various choices of parameters.
- P9.18 Implement a Hopfield network. Train it on digitized patterns of the digits 0–9; study its performance at pattern recall (e.g., of noisy examples of digits) for various resolutions of the patterns.
- P9.19 Design a grammar G that can generate a language $L(G)$ of equilateral triangles of any size; primitives with 0° , 60° , and 120° orientation form the set of terminal symbols $V_t = \{a, b, c\}$.
- P9.20 Design three different grammars producing the language $L(G) = \{ab^n\}$ for $n = 1, 2, \dots$
- P9.21 Design a grammar that generates all characters P or d of the following properties:
- Character P is represented by a square with an edge length equal to one; a vertical line is attached to the bottom left corner of the square and may have any length.
 - Character d is represented by a square with an edge length equal to one; a vertical line is attached to the top right corner of the square and may have any length.

Obviously, there are infinitely many such characters. Use the following set of terminal symbols $V_t = \{N, W, S, E\}$; terminal symbols correspond to directions of the chain code – north, west, south, east. Design your set of non-terminal symbols, using a start symbol s . Validate your grammar design on examples. Show all steps in generating at least two P and two d characters.

- P9.22 Using Algorithm 9.10 prove or disprove isomorphism of the graphs shown in Figure 9.41.

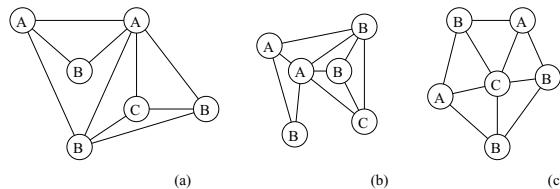


Figure 9.41: Problem P9.22. © Cengage Learning 2015.

- P9.23 Determine the Levenshtein distance for the following string pairs:

- (a) $S_1 = abadcdefacde$ $S_2 = abadddefacde$
 (b) $S_1 = abadcdefacde$ $S_3 = abade faccde$
 (c) $S_1 = abadcdefacde$ $S_4 = cbadcacdae$

- P9.24 Using genetic algorithm optimization, determine the maximum of the following function (this function has several local maxima; its visualization is available in, e.g., Matlab by typing `peaks`, see Figure 9.42):

$$z(x, y) = 3(1 - x)^2 \exp[-x^2 - (y + 1)^2] - 10[(x/5) - x^3 - y^5] \exp[-x^2 - y^2] - (1/3) \exp[-(x + 1)^2 - y^2]$$

Locate or develop a program for genetic algorithm-based optimization following Algorithm 9.12. Design the code string as consisting of n bits for the x value and n bits for the y value, the value of $z(x, y)$ represents the string fitness. Limit your search space to $x \in (-4, 4)$ and $y \in (-4, 4)$. Explore the role of the starting population, population size S , mutation rate M , and string bit length $2n$ on the speed of convergence and solution accuracy. For several values of S, M, n , plot the function values of maximum string fitness, average population fitness, and minimum string fitness as a function of the generation number.

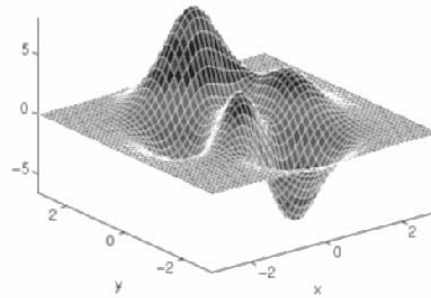


Figure 9.42: Problem P9.24. © Cengage Learning 2015.

- P9.25 Using the definition of the fuzzy set SEVERE as given in Figure 9.30b, sketch the following fuzzy membership functions:
- (a) VERY_SEVERE
 - (b) SOMEWHAT_SEVERE
 - (c) SOMEWHAT_NOT_SEVERE
- P9.26 Using the fuzzy sets DARK and BRIGHT as given in Figure 9.27, sketch the single fuzzy membership function (NOT_VERY_DARK AND NOT_VERY_BRIGHT).
- P9.27 Considering the fuzzy sets A and B given in Figure 9.43, derive intersection, union, and complement of the two fuzzy sets.
- P9.28 Use the composite moments and composite maximum approaches to de-fuzzification to find the representative value of the fuzzy set provided in Figure 9.44.
- P9.29 Flash flood represents a potential danger in many areas, and its prediction is an important part of meteorological forecasting. Clearly, the following conditions increase the flood danger:
- (a) Rain amount in the past three days.
 - (b) Water saturation of soil.
 - (c) The rainfall expected in the next 24 hours.

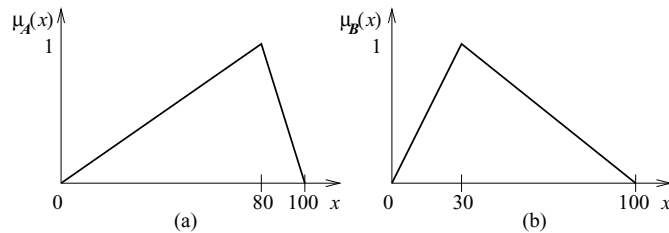


Figure 9.43: Problem P9.27. © Cengage Learning 2015.

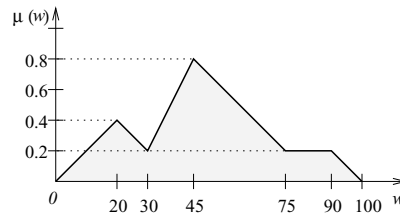


Figure 9.44: Problem P9.28. © Cengage Learning 2015.

Assuming the specified information is available, design a fuzzy logic system to determine the expected flood danger within the next 24 hours.

- P9.30 Develop a program implementation of the fuzzy logic system designed in Problem P9.29. Explore, how different membership function shapes, and fuzzy logic composition and decomposition methods, influence the achieved results.
- P9.31 Using any of the previously developed classifiers as the learning algorithm W , implement Algorithm 9.15 (AdaBoost); train/test this on datasets from earlier Problems, or on publicly available training/testing datasets with 2 classes. Compare AdaBoost performance with that of the other classification approaches.
- P9.32 Using any of the previously developed classifiers to serve as the learning algorithm W , implement Algorithm 9.16 (AdaBoost-MH) and train/test on one of the many publicly available training/testing datasets with more than 2 classes. Compare AdaBoost-MH performance with that of the other classification approaches.
- P9.33 Repeat Problem P9.32 using the random forest approach.
- P9.34 Formulate and solve a practical pattern recognition problem using available open-source packages (e.g., [Weka 3 Team, 2013; R Software Team, 2013]).
- P9.35 Make yourself familiar with solved problems and Matlab implementations of selected algorithms provided in the corresponding chapter of the *Matlab Companion* to this text [Svoboda et al., 2008]. The *Matlab Companion* homepage <http://visionbook.felk.cvut.cz> offers images used in the problems, and well-commented Matlab code is provided for educational purposes.
- P9.36 Use the *Matlab Companion* [Svoboda et al., 2008] to develop solutions to additional exercises and practical problems provided there. Implement your solutions using Matlab or other suitable programming languages.

9.12 References

- Abe S. and Lan M. A method for fuzzy rules extraction directly from numerical data and its application to pattern classification. *IEEE Transactions on Fuzzy Systems*, 3(2):129–139, 1995.
- Aizerman A., Braverman E. M., and Rozoner L. I. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25: 821–837, 1964.
- Ambler A. P. H. A versatile system for computer controlled assembly. *Artificial Intelligence*, 6 (2):129–156, 1975.
- Baird H. S. *Model-Based Image Matching Using Location*. MIT Press, Cambridge, MA, 1984.
- Ballard D. H. and Brown C. M. *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- Barnard S. T. Stereo matching by hierarchical microcanonical annealing. *Perception*, 1:832, 1987.
- Barrero A. Inference of tree grammars using negative samples. *Pattern Recognition*, 24(1):1–8, 1991.
- Barrow H. G. and Popplestone R. J. Relational descriptions in picture processing. *Machine Intelligence*, 6, 1971.
- Beis J. and Lowe D. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the Conference on Computer Vision and Pattern Recognition CVPR*, pages 1000–1006, Puerto Rico, 1997.
- Bishop C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.
- Bollobas B. *Modern Graph Theory*. Springer, Berlin, 2002.
- Boser B. E., Guyon I. M., and Vapnik V. N. A training algorithm for optimal margin classifiers. In *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh PA, 1992. ACM Press.
- Bouman C. and Liu B. Multiple resolution segmentation of textured images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):99–113, 1991.
- Brachman R. and Levesque H. *Knowledge Representation and Reasoning*. Morgan Kaufmann, San Francisco, CA, 2004.
- Breiman L., Friedman J., Olshen R., and Stone C. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
- Breiman L. Random forests. In *Machine Learning*, pages 5–32, 2001.
- Bron C. and Kerbosch J. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
- Buckley F. *Distance in Graphs*. Addison-Wesley, Redwood City, CA, 1990.
- Burges C. J. C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.
- Carpenter G. A. and Grossberg S. ART2: Self organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.
- Cerny V. Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
- Chomsky N. *Syntactic Structures*. Mouton, Hague, 6th edition, 1966.
- Cox E. *The Fuzzy Systems Handbook*. AP Professional, Cambridge, England, 1994.
- Criminisi A. and Shotton J. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Verlag, London, 2013.

- Criminisi A., Shotton J., and Konukoglu E. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Technical Report MSR-TR-2011-114, Microsoft Research, Ltd., Cambridge, UK, 2011.
- Devijver P. A. and Kittler J. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- Duda R. O. and Hart P. E. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
- Duda R. O., Hart P. E., and Stork D. G. *Pattern Classification*. Wiley, New York, 2nd edition, 2000.
- Everitt B. S., Landau S., and Leese M. *Cluster Analysis*. Hodder Arnold, 4th edition, 2001.
- Fischler M. A. and Elschlager R. A. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, C-22(1):67–92, 1973.
- Freund Y. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43: 293–318, 2001.
- Freund Y. and Schapire R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.
- Friedman J., Hastie T., and Tibshirani R. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 38:337–374, 2000.
- Friedman J. H., Bentley J. L., and Finkel R. A. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.
- Fu K. S. *Syntactic Methods in Pattern Recognition*. Academic Press, New York, 1974.
- Fu K. S. *Syntactic Pattern Recognition and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- Garvin M. K., Abramoff M. D., Wu X., Russell S. R., Burns T. L., and Sonka M. Automated 3-D intraretinal layer segmentation of macular spectral-domain optical coherence tomography images. *IEEE Trans. Med. Imag.*, 28:1436–1447, 2009.
- Geman D., Geman S., Graffigne C., and Dong P. Boundary detection by constrained optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):609–628, 1990.
- Goldberg D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- Haeker M., Wu X., Abramoff M. D., Kardon R., and Sonka M. Incorporation of regional information in optimal 3-D graph search with application for intraretinal layer segmentation of optical coherence tomography images. In *Information Processing in Medical Imaging (IPMI)*, volume 4584 of *Lecture Notes in Computer Science*, pages 607–618. Springer, 2007.
- Harary F. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.
- Haupt R. L. and Haupt S. E. *Practical genetic algorithms*. John Wiley & Sons, Inc., New York, NY, 2004.
- Haykin S. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, New York, 2nd edition, 1998.
- Hearst M. Trends and controversies: support vector machines. *IEEE Intelligent Systems*, 13: 18–28, 1998.
- Hecht-Nielsen R. Kolmogorov’s mapping neural network existence theorem. In *Proceedings of the First IEEE International Conference on Neural Networks*, volume 3, pages 11–14, San Diego, 1987. IEEE.

- Ho K. H. L. and Ohnishi N. FEDGE—fuzzy edge detection by fuzzy categorization and classification of edges. In *Fuzzy Logic in Artificial Intelligence. Towards Intelligent Systems. IJCAI '95 Workshop. Selected Papers*, pages 182–196, 1995.
- Homaifar A. and McCormick E. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(2): 129–139, 1995.
- Hopfield J. J. and Tank D. W. Computing with neural circuits: A model. *Science*, 233:625–633, 1986.
- Ishibuchi H., Nozaki K., and Tanaka H. Distributed representation of fuzzy rules and its application to pattern classification. *Fuzzy Sets and Systems*, 52:21–32, 1992.
- Ishibuchi H., Nozaki K., Yamamoto N., and Tanaka H. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3:260–270, 1995.
- Johnson R. A. and Wichern D. W. *Applied Multivariate Statistical Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 1990.
- Kaufman L. and Rousseeuw P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, 1990.
- Kecman V. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. MIT Press, Cambridge, MA, 2001.
- Kirkpatrick S., Gelatt C. D., and Vecchi M. P. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- Kohonen T. *Self-organizing Maps*. Springer Verlag, Berlin; New York, 1995.
- Kolmogorov A. N. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 144:679–681, 1963. (AMS Translation, 28, 55-59).
- Kosko B. *Neural Networks and Fuzzy Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1992.
- Leopold E. and Kindermann J. Text categorization with support vector machines. How to represent texts in input space? *Machine Learning*, 46(1-3):423–444, January 2002.
- Li Y., Bontcheva K., and Cunningham H. Adapting SVM for data sparseness and imbalance: A case study in information extraction. *Natural Language Engineering*, 15(2):241–271, 2009.
- McCulloch W. S. and Pitts W. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- McEliece R. J., Posner E. C., Rodemich E. R., and Venkatesh S. S. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33:461, 1987.
- McHugh J. A. *Algorithmic Graph Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1990.
- Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H., and Teller E. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
- Michalski R. S., Carbonell J. G., and Mitchell T. M. *Machine Learning I, II*. Morgan Kaufmann Publishers, Los Altos, CA, 1983.
- Minsky M. L. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 2nd edition, 1988.
- Mitchell M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.
- Mitchell M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998.

- More J. J. and Toraldo G. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optimization*, 1:93–113, 1993.
- Muja M. and Lowe D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009.
- Niemann H. *Pattern Analysis and Understanding*. Springer Verlag, Berlin–New York–Tokyo, 2nd edition, 1990.
- Nilsson N. J. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, 1982.
- Otten R. H. and van Ginneken L. P. *The Annealing Algorithm*. Kluwer, Norwell, MA, 1989.
- Park W., Hoffman E. A., and Sonka M. Segmentation of intrathoracic airway trees: A fuzzy logic approach. *IEEE Transactions on Medical Imaging*, 17:489–497, 1998.
- Pelleg D. and Moore A. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, 2000. Morgan Kaufmann.
- Pudil P., Novovicova J., and Kittler J. Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125, 1994.
- Quinlan J. R. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
- R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL <http://www.R-project.org>. ISBN 3-900051-07-0.
- R Software Team. The Comprehensive R Archive Network. <http://cran.r-project.org/>, <http://cran.r-project.org/web/packages/e1071/index.html>, 2013.
- Rao C. R. *Linear Statistical Inference and Its Application*. Wiley, New York, 1965.
- Rawlins G. J. E. *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1991.
- Reichgelt H. *Knowledge Representation: An AI Perspective*. Ablex, Norwood, NJ, 1991.
- Rochery M., Schapire R., Rahim M., Gupta N., Riccardi G., Bangalore S., Alshawi H., and Douglas S. Combining prior knowledge and boosting for call classification in spoken language dialogue. In *International Conference on Acoustics, Speech and Signal Processing*, 2002.
- Romesburg C. *Cluster Analysis for Researchers*. Lulu.com, 2004.
- Rosenblatt R. *Principles of Neurodynamics*. Spartan Books, Washington, DC, 1962.
- Rosenfeld A. Fuzzy digital topology. *Information Control*, 40:76–87, 1979.
- Rosenfeld A. The perimeter of a fuzzy set. *Pattern Recognition*, 18:125–130, 1985.
- Rumelhart D. and McClelland J. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986.
- Salvador S. and Chan P. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the 16th IEEE Conference on Tools with Artificial Intelligence, ICTAI-2004*, pages 576–584, Los Alamitos, CA, USA, 2004. IEEE Computer Society.
- Schapire R. E. The boosting approach to machine learning: An overview. In *Proc. MSRI Workshop on Nonlinear Estimation and Classification*, 2002.
- Schlesinger M. I. and Hlaváč V. *Ten lectures on statistical and structural pattern recognition*, volume 24 of *Computational Imaging and Vision*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

- Schoelkopf B. SVMs - a practical consequence of learning theory (part of Hearst, M.A., Trends and controversies: support vector machines). *IEEE Intelligent Systems*, 13:18–21, 1998.
- Sejnowski T. J. and Rosenberg C. R. Parallel systems that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.
- Sharples M., Hogg D., Hutchinson C., Torrance S., and Young D. *Computers and Thought, A Practical Introduction to Artificial Intelligence*. MIT Press, Cambridge, MA, 1989.
- Sklansky J. *Pattern Classifiers and Trainable Machines*. Springer Verlag, New York, 1981.
- Sowa J. F. *Knowledge Representation: Logical, Philosophical, and Computational Foundations: Logical, Philosophical, and Computational Foundations*. Course Technology - Thomson, 1999.
- Svoboda T., Kybic J., and Hlavac V. *Image Processing, Analysis, and Machine Vision: A MATLAB Companion*. Thomson Engineering, 2008.
- Tan H. K., Gelfand S. B., and Delp E. J. A cost minimization approach to edge detection using simulated annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1), 1992.
- Tucker A. *Applied Combinatorics*. Wiley, New York, 3rd edition, 1995.
- Ullmann J. R. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23(1):31–42, 1976.
- van Laarhoven P. J. M. and Aarts E. H. L. *Simulated Annealing: Theory and Applications*. Kluwer and Dordrecht, Norwell, MA, 1987.
- Wechsler H. *Computational Vision*. Academic Press, London–San Diego, 1990.
- Weka 3 Team. Weka 3: Data Mining Software in Java. <http://www.cs.waikato.ac.nz/ml/weka/>, 2013.
- Winston P. H., editor. *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.
- Winston P. H. *Artificial Intelligence*. Addison-Wesley, Reading, MA, 2nd edition, 1984.
- Yang B., Snyder W. E., and Bilbro G. L. Matching oversegmented 3D images to models using association graphs. *Image and Vision Computing*, 7(2):135–143, 1989.
- Young T. Y. and Calvert T. W. *Classification, Estimation, and Pattern Recognition*. American Elsevier, New York–London–Amsterdam, 1974.
- Zadeh L. A. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
- Zdrahal Z. A structural method of scene analysis. In *7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, pages 680–682, 1981.
- Zimmermann H. J., Zadeh L. A., and Gaines B. R. *Fuzzy Sets and Decision Analysis*. North Holland, Amsterdam–New York, 1984.