Chapter 4

Pattern Recognition Concepts

This chapter gives a brief survey of methods used to recognize objects. These methods apply to the recognition of objects in images, but are applicable to any other kind of data as well. The basic approach views an instance to be recognized as a vector of measurements. Several examples are discussed, the central one being the recognition of characters. The reader is also introduced to some simple methods whereby a machine can *learn* to recognize objects by being *taught* from samples. After studying these first four chapters, the reader should understand the design of some complete machine vision systems and should be able to experiment with building a complete set of algorithms for some simple yet real problem.

4.1 Pattern Recognition Problems

In many practical problems, there is a need to make some decision about the content of an image or about the classification of an object that it contains. For example, the user of a notebook computer may be able to give input using handprinted characters. In this case, there would be $m = 128$ ASCII characters and each handprinted object would be classified into one of the m classes. See Figure 4.1. The *classification* of an object – whether it is an 'A' or an '8', etc – would be based on the features of its optical image or perhaps of a pressure footprint, which is also an image-like representation. The classification process might actually fail, either because the character is badly made, or because the person invented a new character. Usually, a *reject class* is included in a system design in order to cover such cases. Image data put into the reject class might be examined again later at some higher level, might result in the formation of a new class, or might just be saved in raw form for viewing.

Imagine an automatic bank teller machine (ATM) using a camera to verify that a current user is indeed authentic. Here, the image of the current person's face is to be matched to a stored image, or images, attached to the current account and stored either on a computer network or in the bank card itself.

1 Definition *The process of matching an object instance to a single object prototype or class definition is called* **verification**.

```
00000000000000000000        00000000000000000000
00000000010000000000        00000000011110000000
00000000110000000000        00000001100001100000
00000000101000000000        00000011000000110000
00000001100110000000        00000100000000010000
00000001000010000000        00001100000000011000
00000010000010000000        00001000000000001000
00000110000001000000        00001000000000011000
00000100000001000000        00001100000000010000
00000100000001100000        00000100000000110000
00001000000000100000        00000111000000100000
00001100111111110000        00000011100111100000
00001111110000010000        00000000111100000000
00011000000000011000        00000011000111000000
00010000000000001000        00000110000001100000
00010000000000001100        00001100000000110000
00110000000000000100        00011000000000011000
00110000000000000110        00110000000000001000
00100000000000000010        00100000000000001100
00100000000000000010        00010000000000011000
01100000000000000010        00011000000000010000
01000000000000000000        00001000000000110000
00000000000000000000        00001110000011100000
00000000000000000000        00000011111110000000
00000000000000000000        00000000000000000000
```

Figure 4.1: Binary images of 'A' and '8'.

In another application, introduced in the exercises of Chapter 1, a food market recognition system would classify fruits and vegetables placed on the checker's scale. The classes would be the set of all identifiable produce items, such as Ida apples, Fuji apples, collard greens, spinach greens, mushrooms, etc., each with a separate name and per pound charge [1].

One definition for *recognition* is *to know again*. A recognition system must contain some memory of the objects that it is to recognize. This memory representation might be built in, perhaps as is the frog's model of a fly; or might be taught by provision of a large number of samples, as a schoolteacher teaches the alphabet; or it might be programmed in terms of specific image features, perhaps as a mother would teach a child to recognize fire trucks versus buses. Recognition and learning of patterns are subjects of considerable depth and interest to cognitive pyschology, pattern recognition, and computer vision. This chapter takes a practical approach and describes methods that have had success in applications, leaving some pointers to the large theoretical literature in the references at the end of the chapter.

[1] Such a system, called Veggie Vision, has already been developed by IBM.

4.2 Common model for classification

We summarize the elements of the *common model of* **classification**: this breakdown is practical rather than theoretical and done so that pattern recognition systems can be designed and built using separately developed hardware and software modules.

Classes

There is a set of m known classes of objects. These are known either by some description or by having a set of examples for each of the classes. For example, for character classification, we have either a description of the appearance of each character or we have a set of samples of each. In the general case, there will be a special *reject class* for objects that cannot be placed in one of the known classes.

2 Definition *An ideal* **class** *is a set of objects having some important common properties: in practice, a class to which an object belongs is denoted by some* **class label**. **Classification** *is a process that assigns a label to an object according to some representation of the object's properties. A* **classifier** *is a device or algorithm which inputs an object representation and outputs a class label.*

3 Definition *A* **reject class** *is a generic class for objects that cannot be placed in any of the designated known classes.*

Sensor/transducer

There must be some device to sense the actual physical object and output a (usually digital) representation of it for processing by machine. Most often, the sensor is selected from existing sensors (off-the-shelf) built for a larger class of problems. For example, to classify vegetables in the supermarket, we could first try using a general color camera that would provide an image representation from which color, shape, and texture features could be obtained. To recognize characters made by an impression using a stylus, we would use a pressure sensitive array.

Since this is a book about machine vision, sensors that produce 2D arrays of sensed data are of most interest. However, pattern recognition itself is more general and just as applicable to recognizing spoken phone numbers, for example, as phone numbers written on paper.

Feature extractor

The feature extractor extracts information relevant to classification from the data input by the sensor. Usually, feature extraction is done in software. Software can be adapted to the sensor hardware on the input side and can evolve through research and development to output results highly relevant to classification. Many image features were defined in the previous chapter.

Classifier

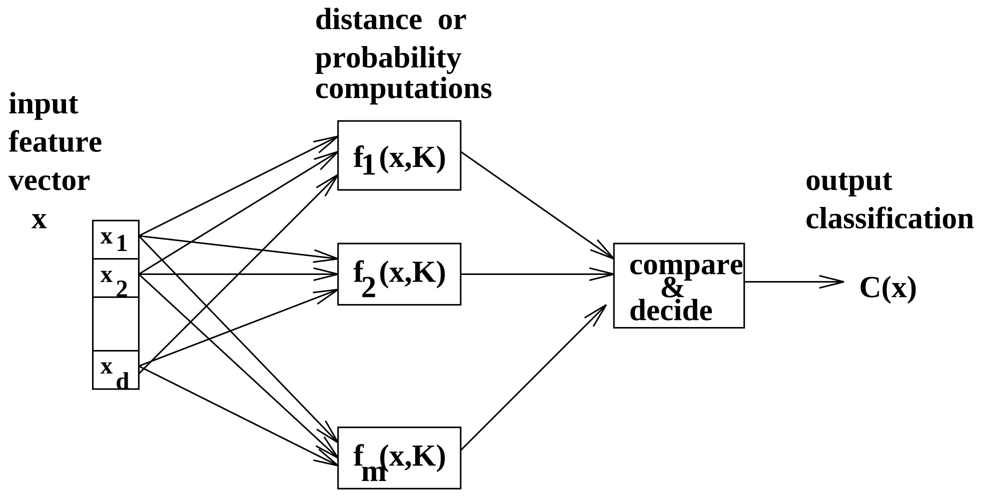


Figure 4.2: Classification system diagram: discriminant functions $f(\mathbf{x}, \mathbf{K})$ perform some computation on input feature vector $\mathbf{x}$ using some knowledge $\mathbf{K}$ from training and pass results to a final stage that determines the class.

The classifier uses the features extracted from the sensed object data to assign the object to one of the m designated classes C_1, C_2, ... , C_{m-1}, $C_m = C_r$, where C_r denotes the reject class.

A block diagram of a classification system is given in Figure 4.2. A $d-$dimensional feature vector $\mathbf{x}$ is the input representing the object to be classified. The system has one block for each possible class, which contains some knowledge $\mathbf{K}$ about the class and some processing capability. Results from the m computations are passed to the final classification stage, which decides the class of the object. The diagram is general enough to model the three different types of classification discussed below: (a) classification using the nearest mean, (b) classification by maximum *a posteriori* probability, and (c) classification using a feed-forward artificial neural network.

Building the classification system

Each of the system parts has many alternative implementations. Image sensors were treated in Chapter 2. Chapter 3 discussed how to compute a number of different features from binary images of objects. Computing color and texture features is treated in Chapters 6 and 7. The character recognition example is again instructive. Characters written in a 30 x 20 window would result in 600 pixels. Feature extraction might process these 600 pixels and output 10 to 30 features on which to base the classification decisions. This example is developed below.

Another common name for the feature extractor is the *preprocessor*. Between the sensor and the classifier, some filtering or noise cleaning must also be performed that should be part of the preprocessing. Some noise cleaning operations were seen in Chapter 3 and more will be studied in Chapter 5. The division of processing between feature extraction and

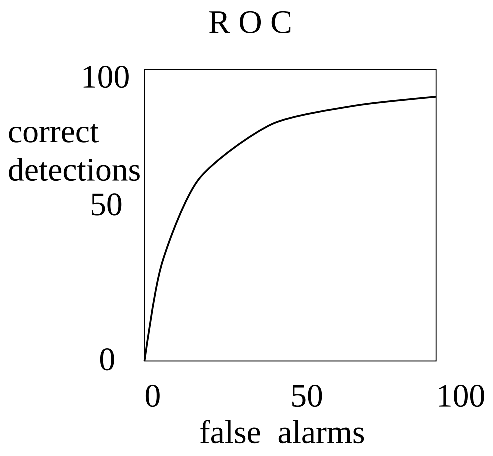


Figure 4.3: The *receiver operating curve* or "ROC" plots correct detection rate versus false alarm rate. Generally, the number of false alarms will go up as the system attempts to detect higher percentages of known objects. Modest detection performance can often be achieved at a low cost in false alarms, but correct detection of nearly all objects will cause a large percentage of unknown objects to be incorrectly classified into some known class.

classification is usually somewhat arbitrary and more due to engineering concerns than to some inherent properties of an application. Indeed, we will look at neural nets, which can do one step classification directly from the input image.

Evaluation of system error

The *error rate* of a classification system is one measure of how well the system solves the problem for which it was designed. Other measures are speed, in terms of how many objects can be processed per unit time, and expense, in terms of hardware, software, and development cost. Performance is determined by both the errors and rejections made; classifying all inputs into the reject class means that the system makes no errors but is useless.

4 Definition (classification error) *The classifier makes a* classification error *whenever it classifes the input object as class C_i when the true class is class C_j; $i \neq j$ and $C_i \neq C_r$, the reject class.*

5 Definition (empirical error rate) *The* empirical error rate *of a classification system is the number of errors made on independent test data divided by the number of classifications attempted.*

6 Definition (empirical reject rate) *The* empirical reject rate *of a classification system is the number of rejects made on independent test data divided by the number of classifications attempted.*

7 Definition (Independent Test Data) *Independent test data are sample objects with true class known, including objects from the "reject class", that were not used in designing*

the feature extraction and classification algorithms.

The above definitions can be used in practice to test the performance of a classification system. We must be very careful to insure that the samples used to design and those used to test the system are representative of the samples that the system will have to process in the future; and, samples used to test the system must be independent of those used to design it. Sometimes, we assume that our data follows some theoretical distribution. With this assumption, we are able to compute a theoretical probability of error for future performance, rather than just an empirical error rate from testing. This concept will be discussed below.

Suppose a handprinted character recognition module for a hand-held computer correctly recognizes 95% of a user's input characters. Given that the user may have to edit an input document anyway, a 5% error rate may be acceptable. Interestingly, such a system might actually train the user, as well as the user training the system, so that performance gradually improves. For example, perhaps the user learns to more carefully close up 8's so that they are not confused with 6's. For a banking system that must read hand-printed digits on deposit slips, a 5% error rate might be intolerable.

False alarms and false dismissals

Some problems are special *two-class problems* where the meaning of the classes might be (a) good object versus bad object, (b) object present in the image versus object absent, or (c) person has disease D versus person does not have disease D. Here, the errors take on special meaning and are not symmetric. Case (c) is most instructive: if the system incorrectly says that the person does have disease D then the error is called a *false alarm or false positive*; whereas, if the system incorrectly says that the person does not have disease D, then the error is called a *false dismissal or false negative*. In the case of the false alarm, it probably means that the person will undergo the cost of more tests, or of taking medicine that is not needed. In the case of false dismissal, the diagnosis is missed and the person will not be treated, possibly leading to grave circumstances. Because the cost of the errors differ greatly, it may make sense to bias the decision in order to minimize false dismissals at the cost of increasing the number of false alarms. Case (a) is less dramatic when the problem is to cull out bruised cherries; a false alarm may mean that the cherry goes into a pie rather than in the produce bin where it would have had more value. False alarms in case (b) may mean we waste energy by turning on a light when there really was no motion in the scene or that we counted an auto on the highway when one really did not pass by: false dismissals in case (b) also have interesting consequences. Figure 4.3 shows a typical *receiver operating curve*, which relates false alarm rate to detection rate. In order to increase the percentage of objects correctly recognized, one usually has to pay a cost of incorrectly passing along objects that should be rejected.

4.3 Precision versus recall

In the application of document retrieval (DR) or image retrieval, the objective is to retrieve *interesting objects* of class C_1 and not too many uninteresting objects of class C_2 according to features supplied in a user's query. For example, the user might be interested in retrieving images of sunsets, or perhaps horses. The performance of such a system is characterized by its *precision* and *recall.*

8 Definition (precision) *The* precision *of a DR system is the number of relevant documents (true C_1) retrieved divided by the total number of documents retrieved (true C_1 plus false alarms actually from C_2).*

9 Definition (recall) *The* recall *of a DR system is the number of relevant documents retrieved by the system divided by the total number of relevant documents in the database. Equivalently, this is the number of true C_1 documents retrieved divided by the total of the true C_1 documents retrieved and the false dismisals.*

For example, suppose an image database contains 200 sunset images that would be of interest to the user and that the user hopes will match the query. Suppose the system retrieves 150 of those 200 relevant images and 100 other images of no interest to the user. The precision of this retrieval (classification) operation is 150/250 = 60% while the recall is 150/200 = 75%. The system could obtain 100% recall if it returned all images in the database, but then its precision would be terrible. Alternatively, if the classification is tightly set for a low false alarm rate, then the precision would be high, but the recall would be low. Image database retrieval will be examined in some detail in Chapter 8.

4.4 Features used for representation

A crucial issue for both theory and practice is *what representation or encoding of the object is used in the recognition process?* Alternatively, what *features* are important for recognition? Let's return to the handprinted character recognition application. Suppose that individual characters can be isolated by a connected components algorithm or by requiring the writer to write them in designated boxes and that we have the following features of each computed by the methods of Chapter 3.

- the *area* of the character in units of black pixels.
- the *height* and *width* of the bounding box of its pixels.
- the *number of holes* inside the character.
- the *number of strokes* forming the character.
- the *center (centroid)* of the set of pixels.
- the *best axis direction* through the pixels as the axis of least inertia.
- the *second moments* of the pixels about the axis of least inertia and most inertia.

Using common-sense reasoning, we can make a table of the properties of characters in terms of these features. The table can be refined by studying the properties of many samples of each character. After doing this, we might have a short decision procedure to classify characters, or at least a set of prototypes to be used for comparison.

Table 4.1 shows 8 features for 10 different characters. For now, assume that there is no error in computing the features. A sequential decision procedure can be used to classify

(class) character	area	height	width	number #holes	number #strokes	(cx,cy) center	best axis	least inertia
'A'	medium	high	3/4	1	3	1/2,2/3	90	medium
'B'	medium	high	3/4	2	1	1/3,1/2	90	large
'8'	medium	high	2/3	2	0	1/2,1/2	90	medium
'0'	medium	high	2/3	1	0	1/2,1/2	90	large
'1'	low	high	1/4	0	1	1/2,1/2	90	low
'W'	high	high	1	0	4	1/2,2/3	90	large
'X'	high	high	3/4	0	2	1/2,1/2	?	large
'*'	medium	low	1/2	0	0	1/2,1/2	?	large
'-'	low	low	2/3	0	1	1/2,1/2	0	low
'/'	low	high	2/3	0	1	1/2,1/2	60	low

Table 4.1: Example features for a sample character set.

instances of these 10 classes as given in Algorithm 1. This structure for classification is called a *decision tree.* Decisions shown in the table are easily implemented in a computer program that has access to the feature values. At each point in the decision process, a small set of features is used to branch to some other points of the decision process; in the current example only one feature is used at each decision point. The branching process models the reduction in the set of possibilities as more features are successively considered.

The current example is used because of its intuitive value. It is naive to suppose that the decision procedure sketched so far is close to a decision procedure that would perform well in a real handprinted character recognition system. Reliably defining and computing the number of strokes, for instance, is very difficult, although we will see a method that might work in Chapter 10. Moreover, methods from Chapters 3 and 5 are needed to remove some of the variations in the data before features are extracted. In some controlled industrial environments, one can set up such simple classification procedures and then adjust the quantitative parameters according to measurements made from sample images. We should expect variations of features within the same class and overlap of some features across classes. Some methods to handle such variation and overlap are studied next.

4.5 Feature Vector Representation

Objects may be compared for similarity based on their representation as a vector of measurements. Suppose that each object is represented by exactly d measurements. The $i-th$ coordinate of such a feature vector has the same meaning for each object A; for example, the first coordinate might be object area, the second coordinate the row moment μ_{rr} defined in the previous chapter, the third coordinate the elongation, and so on. It is convenient for each measurement to be a real or floating point number. The similarity, or closeness, between the the feature vector representations of two objects can then be described using the Euclidean distance between the vectors defined in Equation 4.1. As is shown in Figure 4.4 and discussed in the next section, sometimes the Euclidean distance between an observed vector and a stored class prototype can provide a useful classification function.

input: feature vector with [#holes, #strokes, moment of inertia]
output: class of character

```
 case of #holes

    0: character is 1, W, X, *, -, or /

         case of moment about axis of least inertia

            low: character is  1, -, or /

                   case of best axis direction
                      0: character is  -
                     60: character is  /
                     90: character is  1

          large: character is W or X

                   case of #strokes
                      2: character is X
                      4: character is W

    1: character is A or O

         case of #strokes
            0: character is o
            3: character is A

    2: character is B or 8

         case of #strokes
            0: character is 8
            1: character is B
```

Algorithm 1: Simple decision procedure to classify characters from a set of eight possible characters.

10 DEFINITION *The* **Euclidean distance** *between two d-dimensional feature vectors* $\mathbf{x_1}$ *and* $\mathbf{x_2}$ *is*

$$\| \mathbf{x_1} - \mathbf{x_2} \| = \sqrt{\sum_{i=1,d} (\mathbf{x_1}[i] - \mathbf{x_2}[i])^2} \tag{4.1}$$

4.6 Implementing the Classifier

We now return to the classical paradigm, which represents an unknown object to be classified as a vector of atomic features. A recognition system can be designed in different ways based on feature vectors learned from samples or predicted from models. We examine two alternate methods of using a database of training samples. Assume that there are m classes of objects, not including a reject class, and that we have n_i sample vectors for class i. In our character recognition example from Algorithm 1, we had $m = 10$ classes of characters; perhaps we would have $n_i = 100$ samples from each. The feature vectors have dimension $d = 8$ in this case.

Classification using the nearest class mean

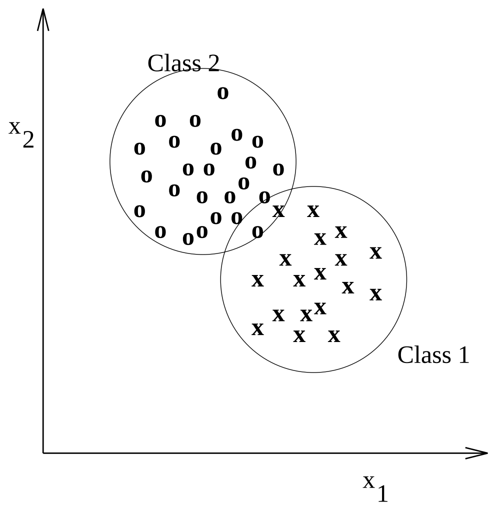


Figure 4.4: Two compact classes: classification using nearest mean will yield a low error rate.

A simple classification algorithm is to summarize the sample data from each class using the class mean vector, or *centroid*, $\mathbf{\bar{x}_i} = 1/n_i \sum_{j=1,n_i} \mathbf{x}_{i,j}$ where $\mathbf{x}_{i,j}$ is the $j - th$ sample feature vector from class i. An unknown object with feature vector $\mathbf{x}$ is classified as class i if it is [much] closer to the mean vector of class i than to any other class mean vector. We have the opportunity to put $\mathbf{x}$ into the reject class if it is not close enough to any of the sample means. This classification method is simple and fast and will work in some problems where the sample vectors from each class are compact and far from those of the other classes. A simple two class example with feature vectors of dimension $d = 2$ is shown in Figure 4.4:

sample vectors of class one are denoted by **x** and those of class two are denoted by **o**. Since there are samples of each class that are equidistant from both class centroids, the error rate will not be zero, although we expect it to be very low if the structure of the samples well represents the structure of future sensed objects. We now have one concrete interpretation for the function boxes of Figure 4.2: the $i-$th function box computes the distance between unknown input **x** and the mean vector of training samples from that class. The training samples constitute the knowledge **K** about the class.

Exercise 1 classifying coins

Obtain 10 samples of each coin used in the U.S. (penny, nickle, dime, quarter, half-dollar, dollar). Using a micrometer, measure the diameter and thickness of each of the 60 samples to the nearest 0.01 in. and then create a scatter plot of the six classes as done in Figure 4.4. (Always measure the thickness either at the center or across the edge.) Estimate the error rate of a classifier based on the nearest mean computation.

Difficulties can arise when the structure of class samples is complex. Figure 4.5 shows a case where class samples are well separated, but the structure is such that nearest mean classification will yield poor results for multiple reasons. First of all, class two (**o**) is *multi-modal*: its samples lie in two separate compact clusters that are not represented well by the overall mean that lies midway between the two modes. Several of the samples from class one (**x**) are closer to the mean of class two than to the mean of class one. By studying the samples, we might discover the two modes of class two and be able to separate class two into two subclasses represented by two separate means. While this is simple using a 2D scatter plot as in Figure 4.5, it may be very difficult to understand the structure of samples when dimension d is much higher than 2. A second problem is due to the elongation of classes one and three. Clearly, samples of class three with large coordinate x_2 are closer to the mean of class two than they are to the mean of class three. Similarly, samples of class one (**x**) with small coordinate x_1 will still be closer to the mean of one of the subclasses of class two, even if class two is split into two modes. This problem can be reduced by modifying the distance computation to take into consideration the different spread of the samples along the different dimensions.

We can compute a modified distance **from** unknown feature vector **x to** class mean vector $\mathbf{x_c}$ by scaling by the spread, or *standard deviation*, σ_i of class c along each dimension i. The standard deviation is the square root of the variance.

11 Definition **scaled Euclidean distance from x to class mean $\mathbf{x_c}$:**

$$|| \ \mathbf{x} \ - \ \mathbf{x_c} \ || \ = \ \sqrt{\sum_{i=1,d} ((\mathbf{x}[i] - \mathbf{x_c}[i])/\sigma_i)^2} \qquad (4.2)$$

Scaling is almost always required due to the different units along the different dimensions. For example, suppose we were classifying vehicles using features $\mathbf{x}[1]$ = length in feet and $\mathbf{x}[2]$ = weight in pounds: without scaling, the Euclidean distance would be dominated by the large numbers of pounds, which would overshadow any discrimination due to vehicle

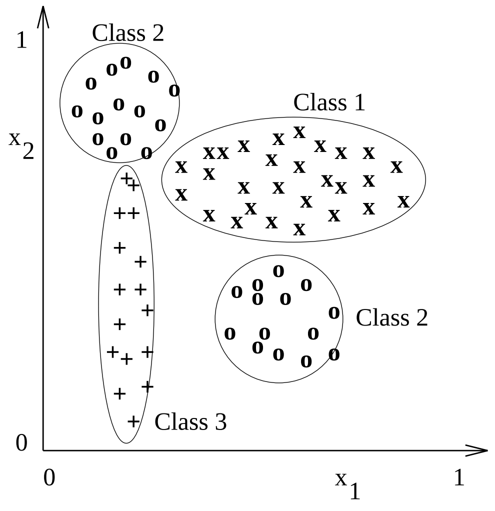


Figure 4.5: Three classes with complex structure: classification using nearest mean will yield poor results.

length.

In the case shown in Figure 4.5, with separate mean vectors for the two modes of class two, such separate class-dependent scaling of the features x_1 and x_2 would yield good classification results. Most cases are not so easy, however. If the ellipses representing the distribution of class samples are not aligned with the coordinate axes, as they are in Figure 4.5, then coordinate transformations are needed in order to properly compute the distance of an unknown sample to a class mean. This is discussed below under the Bayesian classification scheme. A harder problem results if the set of samples has a curved structure in d-dimensional space.

Classification using the Nearest Neighbors

A more flexible but more expensive method of classification is to classify unknown feature vector $\mathbf{x}$ into the class of the individual sample closest to it. This is the *nearest neighbor rule*. Nearest neighbor classification can be effective even when classes have complex structure in $d-$space and when classes overlap. No assumptions need to be made about models for the distribution of feature vectors in space; the algorithm uses only the existing training samples. A brute force approach (algorithm below) computes the distance from $\mathbf{x}$ to all samples in the database of samples and remembers the minimum distance. One advantage of this approach is that new labeled samples can be added to the database at any time. There are data structures that can be used to eliminate many unnecessary distance computations. Tree-structured or gridded data sets are two examples that are described in the chapter references.

A better classification decision can be made by examining the nearest k feature vectors in the database. $k > 1$ allows a better sampling of the distribution of vectors in d-space:

this is especially helpful in regions where classes overlap. It has been shown that in the limit as the number of samples grows to infinity, the error rate for even $k = 1$ is no worse than twice the optimal error rate. In theory, we should do better with $k > 1$; but, effectively using a larger k depends on having a larger number of samples in each neighborhood of the space to prevent us from having to search too far from $\mathbf{x}$ for samples. In a two-class problem using $k = 3$, we would classify $\mathbf{x}$ into the class that has 2 of the 3 samples nearest $\mathbf{x}$. If there are more than two classes then there are more combinations possible and the decision is more complex. The algorithm given below classifies the input vector into the reject class if there is no majority of the nearest k samples from any one class. This algorithm assumes no structure on the set of training samples; without such structure, the algorithm becomes slower with increasing number of samples n and k. Algorithms that use efficient data structures for the samples can be found via the references at the end of the chapter.

S is a set of n labeled class samples s_i where $s_i.\mathbf{x}$ is a feature vector and $s_i.c$ is its integer class label.
$\mathbf{x}$ is the unknown input feature vector to be classified.
A is an array capable of holding up to k samples in sorted order by distance d.
The value returned is a class label in the range $[\mathbf{1}, \mathbf{m}]$

```
procedure K_Nearest_Neighbors($\mathbf{x}, \mathbf{S}$)
{
make A empty;
for all samples $s_i$ in S
{
  $d$ = Euclidean distance between $s_i$ and $\mathbf{x}$;
  if A has less than $k$ elements then insert $(d, s_i)$ into A;
  else if $d$ is less than max A
    then {
            remove the max from A;
            insert $(d, s_i)$ in A;
         }
} ;
assert A has $k$ samples from S closest to $\mathbf{x}$;
if a majority of the labels $s_i.c$ from A are class $c_0$
  then classify $\mathbf{x}$ into class $c_o$;
  else classify $\mathbf{x}$ into the reject class;
return(class_of_x);
}
```

Algorithm 2: Compute the K-Nearest Neighbors of x and return majority class.

4.7 Structural Techniques

Simple numeric or symbolic features of an object may not be sufficient for recognition. For example, consider the two characters shown in Figure 4.6. They have identical bounding boxes, the same numbers of holes and strokes, the same centroid, and the same second moments in the row and column directions, and their major axis directions are within .1

radian of being the same. Each of these characters has two *bays*, which are intrusions of the background into the character. Each bay has a *lid*, a virtual line segment that closes up the bay. The main differentiating feature of these two characters is a *relationship*, the spatial relationship between these two bays. In the character on the left, the lid of the upper bay is to the right of the lid of the lower bay. In the character on the right, the lid of the upper bay is to the left of the lid of the lower bay. This suggests that relationships among primitive features can be used as higher-level and potentially more powerful features for recognition. The field of *structural pattern recognition* has developed from this premise.

0	0	0	0	0	0	0	0	0	0
0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	**1**	0	0	0	0	0	0	**1**	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	**1**	**1**	0
0	**1**	0	0	0	**1**	**1**	**1**	0	0
0	0	**1**	**1**	**1**	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0
0	**1**	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	**1**	0	0	0	0	0	0	0
0	**1**	0	0	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0	0
0	0	**1**	**1**	**1**	0	0	0	**1**	0
0	0	0	0	**1**	**1**	**1**	**1**	0	0
0	0	0	0	0	0	0	0	0	0

Figure 4.6: Two characters that have the same global features but a different structure.

Statistical pattern recognition traditionally represents entities by feature vectors, which are typically vectors of atomic values, such as numbers and Boolean values (T or F). These values measure some global aspect of the entities, such as area or spatial moments. Our character example goes one step further in that it measures the number of holes and the number of strokes in each character. This implies the existance of a hole-finding procedure to find and count the holes and of some type of segmentation algorithm that can partition the character into strokes.

In structural pattern recognition, an entity is represented by its primitive parts, their attributes, and their relationships, as well as by its global features. Figure 4.7 illustrates three separate letter A's that have approximately the same structure. Each can be broken up into 4 major strokes: two horizontal and two vertical or slanted. Each has a hole or 'lake' near the top of the character with a bay below it; the lake and the bay are separated by a horizontal stroke.

When the relationships among primitives are binary relations, a structural description of an entity can be viewed as a graph structure. Suppose the following relationships over strokes, bays, and lakes are useful in recognizing characters:

- **CON:** specifies the connection of two strokes
- **ADJ:** specifies that a stroke region is immediately adjacent to a lake or bay region

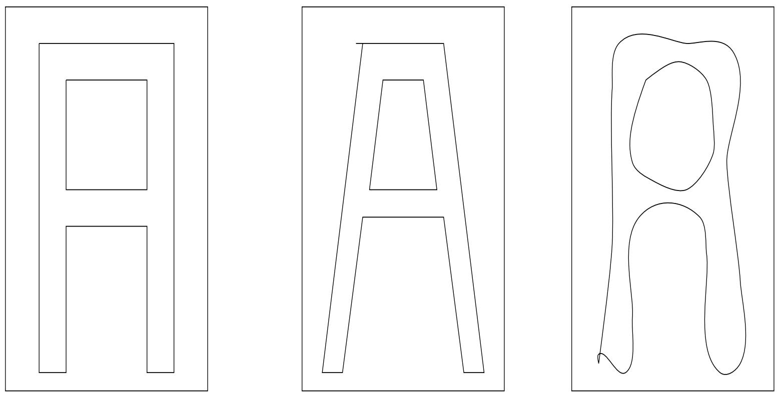


Figure 4.7: Three A's with similar structure.

- **ABOVE:** specifies that one hole (lake or bay) lies above another

Figure 4.8 shows a graph representation of the structural description of the character 'A' using these three binary relations. Higher-level relations, ie. ternary or even quaternary, can be used if they can be defined to provide even stronger constraints. For example, a ternary relationship exists among the lake, the horizontal stroke below it, and the bay below the stroke.

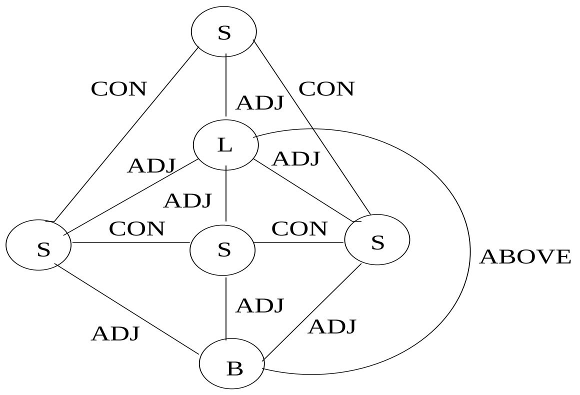


Figure 4.8: A graph structure representing the letter 'A'. 'S','L','B' denote *side, lake, bay* respectively.

Structural pattern recognition is often achieved through *graph matching* algorithms, which will be covered in Chapter 14. However, the relationship between two primitives can itself be considered an atomic feature and thus can be used in a feature vector and incorporated into a statistical decision procedure. One simple way to do this is to merely count the number of times a particular relationship between two particular feature types (ie. a bay

		class j output by the pattern recognition system										
		'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'	'R'
	'0'	97	0	0	0	0	0	1	0	0	1	1
	'1'	0	98	0	0	1	0	0	1	0	0	0
true	'2'	0	0	96	1	0	1	0	1	0	0	1
object	'3'	0	0	2	95	0	1	0	0	1	0	1
class	'4'	0	0	0	0	98	0	0	0	0	2	0
	'5'	0	0	0	1	0	97	0	0	0	0	2
i	'6'	1	0	0	0	0	1	98	0	0	0	0
	'7'	0	0	1	0	0	0	0	98	0	0	1
	'8'	0	0	0	1	0	0	1	0	96	1	1
	'9'	1	0	0	0	3	0	0	0	1	95	0

Figure 4.9: Hypothetical confusion matrix for digit recognition. 'R' is the reject class.

beneath a horizontal stroke) appears in a pattern. The integer value of the count becomes a feature for recognition of the overall pattern.

Structural methods are useful for recognition of complex patterns involving many subpatterns. They also offer advantages in higher-level understanding of a scene, especially when multiple objects are present. From one general viewpoint, structural pattern recognition together with the other methods of this chapter emcompasses all of computer vision: within this view, the remaining chapters of the book can be taken to provide more methods of extracting features and parts from 2D or 3D objects and scenes.

4.8 The Confusion Matrix

12 Definition *The* confusion matrix *is commonly used to report results of classification experiments. Figure 4.9 gives an example. The entry in row i, column j records the number of times that an object labeled to be truly of class i was classified as class j.*

The confusion matrix diagonal, where $i = j$, indicates the successes: with perfect classification results, all off-diagonal elements are zero. High off-diagonal numbers indicate confusion between classes and force us to reconsider our feature extraction procedures and/or our classification procedure. If we have done thorough testing, the matrix indicates the kinds and rates of errors we expect in a working system. In the example shown in Figure 4.9, 7 of 1000 vectors input to the system were rejected. Three inputs labeled as 9 were incorrectly classified as 4, while two inputs labeled as 4 were incorrectly classified as 9. Altogether, 25 of the input vectors were misclassified. Assuming that the test data was independent of that used to train the classification system, we would have an empirical reject rate of $7/1000 = 0.007$ and an (overall) error rate of $25/1000 = 0.025$. The error rate for just 9s, however, is $5/100 = 0.05$.

4.9 Decision Trees

When a pattern recognition task is complex and involves many different potential features, comparing an entire unknown feature vector to many different pattern feature vectors may be too time consuming. It may even be impossible as in the case of medical diagnosis, where measurement of a feature usually means a costly and perhaps painful lab test. Use of a decision tree allows feature extraction and classification steps to be interleaved. The decision tree is a compact structure that uses one feature (or, perhaps a few features) at a time to split the search space of all possible patterns. The simple decision procedure of Algorithm 1 implements the flow of control shown in the decision tree of Figure 4.10. This tree has nodes that represent different features of the feature vector. Each branching node has one child per possible value for its feature. The decision procedure selects a child node based on the value of the specified feature in the unknown feature vector. A child node may specify another feature to be tested, or it may be a leaf node containing the classification associated with that path through the tree.

13 Definition *A binary decision tree is a binary tree structure that has a decision function associated with each node. The decision function is applied to the unknown feature vector and determines whether the next node to be visited is the left child or the right child of the current node.*

In the simplest case with numeric feature values, the decision function at a node merely compares the value of a particular feature of the unknown feature vector to a threshold and selects the left child if the value of the feature is less than the threshold and the right child otherwise. In this case, only the feature to be used and the threshold value need to be stored in each branch node of the tree. Each leaf node stores the name of a pattern class; if the decision tree procedure reaches a leaf node, the unknown feature vector is classified as belonging to that pattern class. Figure 4.11 illustrates this type of decision tree, which was constructed to correctly classify the training data shown.

The tree of Figure 4.11 was constructed manually by looking at the data and picking suitable features and thresholds. The training data here is just a toy example; real data is likely to have many more features and many more samples. It is not uncommon to have several hundred features and thousands of training samples for real applications such as medical diagnosis. In this case, an automated procedure for producing decision trees is needed. Furthermore, for any given set of training samples, there may be more than one decision tree that can classify them. So it is important to select features that give the best decision tree by some criterion. Usually a tree that is simpler or one that has fewer levels and therefore fewer tests is preferred.

Consider the training data and two possible decision trees shown in Figure 4.12. Both trees can discriminate between the two classes, class I and class II, shown in the training data. The tree on the left is very simple; it is able to classify a feature vector with only a single comparison. The tree on the right is larger and requires more comparisons.

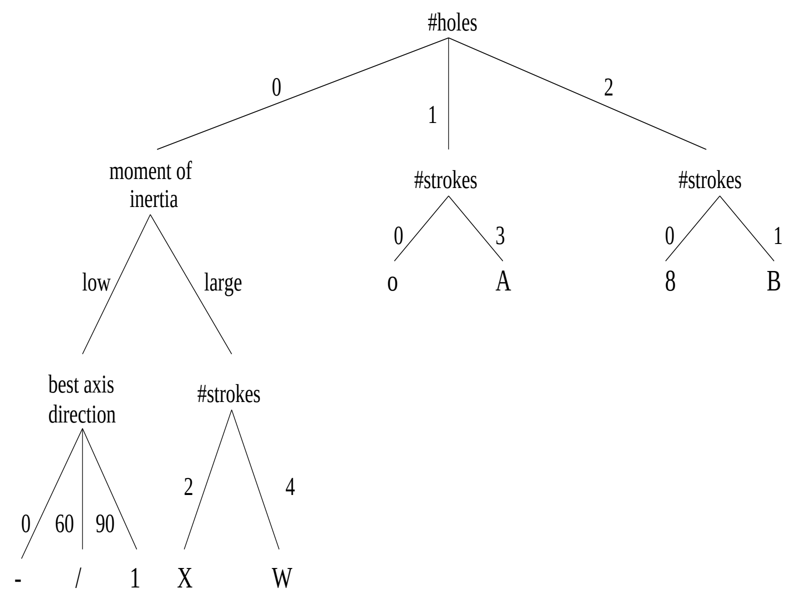
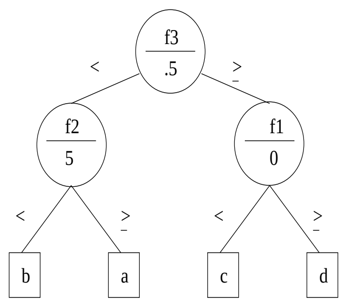


Figure 4.10: Decision tree that implements the classification procedure of Figure 1

f1	f2	f3	CLASS
3	6	0	a
-5	9	1	c
4	5	1	d
7	4	0	b
-1	10	0	a
2	6	1	d
-2	2	1	c
-1	3	0	b

Training Data

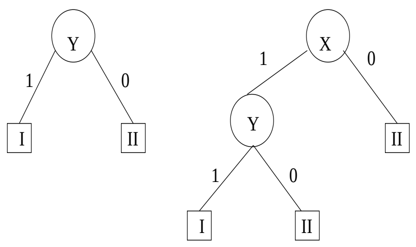


Decision Tree

Figure 4.11: Binary decision tree based on a feature and threshold value at each node.

X	Y	Z	Class C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Training Data



Two Possible Decision Trees

Figure 4.12: Two different decision trees that can each classify the given training samples.

Automatic Construction of a Decision Tree

There are a number of different methods for constructing optimal decision trees from training data, each with its own definition of optimality. (See Haralick and Shapiro, Vol. I, Chapter 4 for an in-depth coverage.) One simple, but effective method is grounded in information theory. The most basic concept in information theory is called *entropy.*

14 Definition *The* **entropy of a set of events** $x = \{x_1, x_2, \ldots, x_n\}$ *where each* x_i *is an event is*

$$H(x) = -\sum_{i=1}^{n} p_i log_2 p_i \tag{4.3}$$

where p_i *is the probability of event* x_i*.*

Entropy can be interpreted as the average uncertainty of the information source. Quinlan(1986) used an entropy-based measure called information gain to evaluate features and produce optimal decision trees.

Examples of entropy computations for sets of possible events

Consider the set of three possible events with their associated probabilities.

$$X = \{(x_1,\ 3/4),\ (x_2,\ 1/8),\ (x_3,\ 1/8)\}$$

The computation of entropy is as follows:

$$\begin{aligned} H(x) &= -[(3/4)log_2(3/4)\ +\ (1/8)log_2(1/8)\ +\ (1/8)log_2(1/8)] \\ &= -[(3/4)(-0.415)\ +\ (1/8)(-3)\ +\ (1/8)(-3)] \\ &= 1.06 \end{aligned}$$

Similarly, a set of four equally likely events has entropy 2.0.

$$\begin{aligned} X &= \{(x_1,\ 1/4),\ (x_2,\ 1/4),\ (x_3,\ 1/4)\ (x_4,\ 1/4)\} \\ H(x) &= -[4\ ((1/4)(-2))]\ =\ 2 \end{aligned}$$

Exercise 2

(a) Compute the entropy of a set of two equally likely events. (b) Compute the entropy of a set of four possible outcomes with probabilities $\{1/8,\ 3/4,\ 1/16,\ 1/16\}$.

Information theory allows us to measure the *information content* of an event. In particular, the information content of a class event with respect to each of the feature events is useful for our problem. The information content $I(C;F)$ of the class variable C with possible values $\{c_1, c_2, \ldots, c_m\}$ with respect to the feature variable F with possible values $\{f_1, f_2, \ldots, f_d\}$ is defined by

$$I(C;F) = \sum_{i=1}^{m}\sum_{j=1}^{d} P(C=c_i, F=f_j) log_2 \frac{P(C=c_i, F=f_j)}{P(C=c_i)P(F=f_j)} \tag{4.4}$$

where $P(C=c_i)$ is the probability of class C having value c_i, $P(F=f_j)$ is the probability of feature F having value f_j, and $P(C=c_i, F=f_j)$ is the joint probability of class $C=c_i$ and variable $F=f_j$. These prior probabilities can be estimated from the frequency of the associated events in the training data. For example, since class I occurs in two out of the four training samples (see Figure 4.12), $P(C=I) = 2/4 = .5$. Since three of the four training samples have value 1 for feature X, $P(X=1) = 3/4 = .75$.

We can use this information content measure to decide which feature is the best one to select at the root of the tree. We calculate $I(C,F)$ for each of the three features: X, Y, and Z.

$$\begin{aligned} I(C,X) &= P(C=I, X=1) log_2 \frac{P(C=I, X=1)}{P(C=I)P(X=1)} \\ &+ P(C=I, X=0) log_2 \frac{P(C=I, X=0)}{P(C=I)P(X=0)} \\ &+ P(C=II, X=1) log_2 \frac{P(C=II, X=1)}{P(C=II)P(X=1)} \end{aligned}$$

$$
\begin{aligned}
&+ \quad P(C=II, X=0) log_2 \frac{P(C=II, X=0)}{P(C=II)P(X=0)} \\
&= \quad .5 log_2 \frac{.5}{.5 \times .75} + 0 + .25 log_2 \frac{.25}{.5 \times .25} + .25 log_2 \frac{.25}{.5 \times .75} \\
&= \quad 0.311
\end{aligned}
$$

$$
\begin{aligned}
I(C,Y) &= .5 log_2 \frac{.5}{.5 \times .5} + 0 + .5 log_2 \frac{.5}{.5 \times .5} + 0 \\
&= 1.0
\end{aligned}
$$

$$
\begin{aligned}
I(C,Z) &= .25 log_2 \frac{.25}{.5 \times .5} + .25 log_2 \frac{.25}{.5 \times .5} + .25 log_2 \frac{.25}{.5 \times .5} + .25 log_2 \frac{.25}{.5 \times .5} \\
&= 0.0
\end{aligned}
$$

Feature Y, which has an information content of 1.0, gives the most information in determining class and so should be selected as the first feature to be tested at the root node of the decision tree. In the case of this simple example, the two classes are completely discriminated and the tree is complete with a single branch node. In the more general case, at any branch node of the tree when the selected feature does not completely separate a set of training samples into the proper classes, the set of samples is partitioned according to the decision procedure at that node and the tree construction algorithm is invoked recursively for the subsets of training samples at each of the child nodes for which more than one class is still present.

The algorithm described here was meant to operate on a decision tree like the one of Figure 4.10, a general tree that has branches for each possible value of the feature being tested at a node. In order to adapt it to the binary threshold type tree like the one of Figure 4.11, the information content of each feature-threshold pair would have to be considered for each possible threshold. Although this sounds like an infinite set of possibilities, only a finite number of values for each feature appear in the training data, and this finite set is all that need be considered.

The above example is very simple, but it is possible to automatically construct real, useful decision trees on tens or even hundreds of features. Consider again the character recognition problem, but this time for more difficult, hand-printed characters. Some useful features for this type of characters are *lakes*, *bays*, and *lids*, as discussed in Section 4.6. Lakes are holes in the character (regions of label 0 completely surrounded by character pixels of label 1), bays are intrusions into the character (regions of label 0 that are only partially surrounded by character pixels of label 1), and lids are segments that can be used to close up the bays. Figure 4.13 shows a hand-printed character six (a), its bay and lake features (b), and its lid feature (c). The operations of mathematical morphology described in Chapter 3 can be used to extract these useful primitive features. From them, the following numeric features can be computed:

- **lake_num:** the number of lakes extracted
- **bay_num:** the number of bays extracted

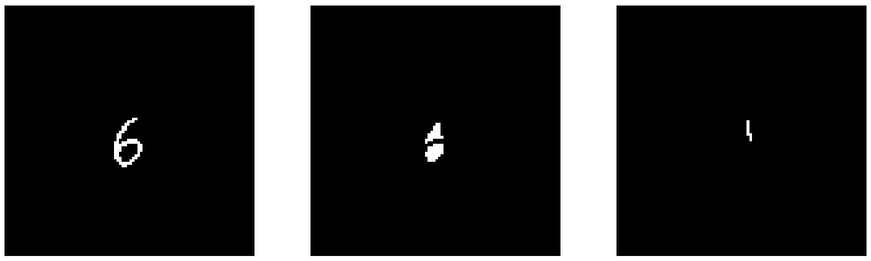


Figure 4.13: a) Image of a hand-printed character six; b) the bay (above) and lake (below) extracted by morphological image processing; c) the lid of the bay extracted by further morphological processing.

- **lid_num:** the number of lids extracted
- **bay_above_bay:** Boolean feature that is true if any bay is completely above another
- **lid_rightof_bay:** Boolean feature that is true if any lid is completely to the right of a bay
- **bay_above_lake:** Boolean feature that is true if any bay is completely above a lake
- **lid_bottom_of_image:** Boolean feature that is true if the lowest point of any lid is within a few pixels of the lowest point of the whole character

With sufficient training data, these features can be used to construct a decision tree that can classify the hand-printed numeric digits. Figure 4.14 illustrates a sample set of training data for the digits zero through nine.

Exercise 3 Decision tree construction

Given the training data shown in Figure 4.14 write a program that uses information content to construct a decision tree to discriminate among the ten digit classes. How well does the tree constructed on all 40 samples work on the training data? What happens if you construct the tree from the last 20 samples and then test it on the first 20 samples?

Exercise 4

(a) Describe how to extract a lake using morphological image processing from Chapter 3.
(b) Describe how to extract a lid, given that a bay has already been identified.

4.10 Bayesian decision-making

We examine how the knowledge of probability distributions can be used to make classification decisions with least expected error rate. Suppose we take a single measurement x

lake_num	bay_num	lid_num	bay_above_bay	lid_rightof_bay	bay_above_lake	lid_bottomof_image	class
1	0	0	F	F	F	F	0
1	0	0	F	F	F	F	0
0	0	0	F	F	F	F	1
0	2	2	F	T	F	T	2
0	2	2	T	F	F	F	3
1	1	1	F	F	F	T	4
1	1	1	F	T	T	F	6
0	2	2	T	T	F	F	2
0	2	2	T	T	F	T	4
0	1	1	F	F	F	F	7
0	0	0	F	F	F	F	1
1	0	0	F	F	F	F	0
1	1	1	F	F	F	F	9
0	2	2	T	T	F	F	2
1	1	1	F	F	F	F	9
0	1	1	F	F	F	F	1
0	2	2	T	F	F	T	4
0	2	2	T	T	F	F	5
1	1	1	F	T	T	F	6
0	2	2	T	F	F	F	3
0	1	1	F	F	F	F	1
1	0	0	F	F	F	F	0
0	2	2	T	T	F	F	5
1	1	1	F	T	T	F	6
0	1	1	F	F	F	T	7
2	0	0	F	F	F	F	8
1	1	1	F	F	F	F	9
1	0	0	F	F	F	F	0
2	0	0	F	F	F	F	8
1	1	1	F	T	T	F	6
0	2	2	F	T	F	F	7
1	1	1	F	F	F	F	9
1	0	0	F	F	F	F	0
0	2	2	T	F	F	F	3
0	2	2	T	T	F	F	5
0	2	2	T	T	F	F	2
0	2	2	T	T	F	F	2
0	1	1	F	F	F	F	1
0	1	1	F	F	F	F	7
0	2	2	T	T	F	F	5
0	2	2	T	F	F	T	4
0	2	2	T	F	F	F	3

Figure 4.14: Training data for hand-printed characters.

from an infrared image of a dark red cherry and use it to determine whether the cherry is bruised or not. An unbruised cherry is in class ω_1, while a bruised cherry is in class ω_2. Also, suppose that we have studied a very large number of surface elements from a large number of bruised and unbruised cherries, so that we have the knowledge captured in the distribution functions shown in Figure 4.15. The curve at the right, $p(x|\omega_1)$, shows the distribution of measurement x over a large number of surface samples of unbruised cherries. The curve on the left, $p(x|\omega_2)$, shows the distribution of measurement x over a large number of bruised surface samples. The data has been normalized so that the area under each curve is 1.0, making each a probability distribution. (Bruised tissue contains water, which absorbs infrared radiation more than unbruised tissue and thus low reflectance is much more likely for those cherries. The water content varies and so does the darkness of the skin color, causing the distributions to overlap – some dark unbruised cherries will reflect similar to some bright bruised cherries.)

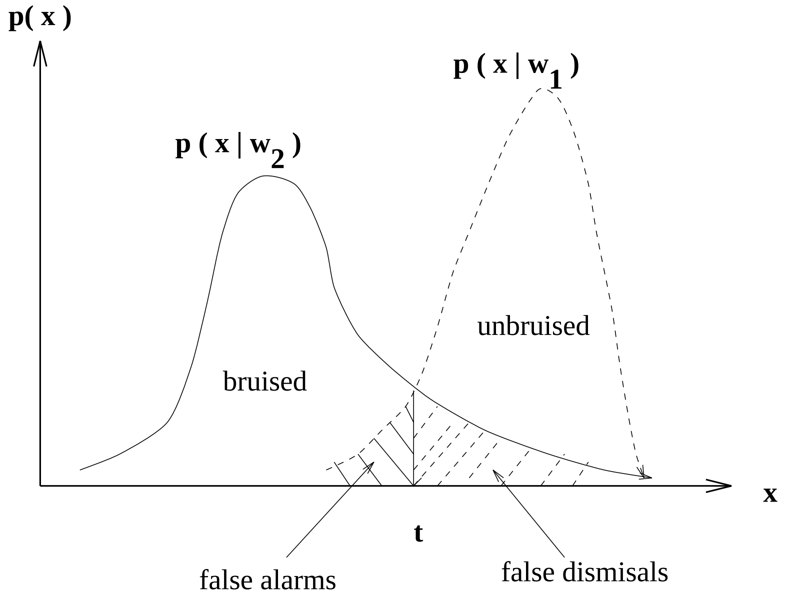


Figure 4.15: Distributions for intensity measurement x conditioned on whether x is taken from an unbruised or bruised cherry.

If bruised cherries are as likely to occur as unbruised ones, and if all classification errors cost us the same, then we can make the decision ω_1 whenever $x > t$ and ω_2 otherwise. For such a decision policy, the hatched area to the right of t represents (twice) the false dismisal rate: this is the probability of getting the acceptably high measurement x from a bruised cherry. The area is twice the false dismissal rate because of the assumption that the *a priori* probability for each density is 0.5, and thus each density should be scaled down so that the total area under the curve is 0.5. The hatched area to the left of t represents (twice) the false alarm probability: this is the probability that a good cherry will be classified as bruised because $x < t$. Because bruised and unbruised cherries are assumed to be equally likely to occur as input to our system, each curve would actually represent only 0.5 of the overall probability so all areas are shown to be twice their actual size. The total error is the total hatched area under the curves. It is important to observe that moving decision threshold t either to the left or to the right will result in a **greater** hatched area and hence a greater error.

The example above considered only the special case of two equally likely classes with errors of equal cost. Now we extend the method to cover the case of m classes all with possibly different *a priori* probabilities. We keep the simplifying assumption that all errors are equally costly. Using Bayesian decision-making, we classify an object into the class to which it is most likely to belong.

15 DEFINITION *A* **Bayesian classifier** *classifies an object into the class to which it is most likely to belong based on the observed features.*

In order to compute the likelihoods given the measurement x, the following distributions are needed.

$$\textbf{class conditional distribution}: \quad p(x|\omega_i) \ \textit{for each class}\ \omega_i \tag{4.5}$$

$$\textbf{a priori probability}: \quad P(\omega_i) \ \textit{for each class}\ \omega_i \tag{4.6}$$

$$\textbf{unconditional distribution}: \quad p(x) \tag{4.7}$$

If all of the classes ω_i are disjoint possibilities covering all possible cases, we can apply Bayes' rule to compute *a posteriori* probabilities for each of the classes given the *a priori* probabilities of the classes and the distributions of x for each class.

$$P(\omega_i|x) \ = \ \frac{p(x|\omega_i)P(\omega_i)}{p(x)} \ = \ \frac{p(x|\omega_i)P(\omega_i)}{\sum_{i=1,m} p(x|\omega_i)P(\omega_i)} \tag{4.8}$$

Returning to the classifier sketched in Figure 4.2, inside each of the class computation boxes we make $f_i(x, K) = P(\omega_i|x)$, which by Bayes Rule in Equation 4.8 can be computed as $p(x|\omega_i)P(\omega_i)/p(x)$. Since $p(x)$ is the same for all the class computation boxes, we can ignore it and just make the classification decision ω_i for the maximum $p(x|\omega_i)P(\omega_i)$. To design our Bayes classifier, we must have as knowledge K the *prior* probability of each class $P(\omega_i)$ and the class conditional distribution of $p(x|\omega_i)$. Having such knowledge allows us to design for optimal future decisions. It is often difficult to establish these *prior* probabilities. For example, how would we know the probability that an arbitrary cherry entering our sorting machine is a bruised cherry? If this varies with the weather and picking crews, it make take too much sampling work to obtain the needed information whenever conditions change.

Parametric Models for Distributions

In practice, we must implement the computation of $p(x|\omega_i)$ in some manner. An empirical method is to quantize the range of x and record the frequency of occurences of x in the samples for each interval and store the result in an array or histogram. One could fit a smooth spline function to this data to produce a probability function valid for all real x. Note that we need to scale our results so that the sum over all possible values of x is 1.0. If we observe that the distribution of x follows some known parametric model, then we can represent the distribution by the small number of parameters that characterize it. Poisson, exponential, and normal (or Gaussian) distributions are commonly used. A normal distribution is the well-known "bell-shaped curve" sometimes used to assign grades in college courses.

16 Definition *A* **normal distribution** *characterized by mean μ and standard deviation σ is defined as follows.*

$$p(x) \;=\; N(\mu, \sigma)(x) \;=\; \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{1}{2}(\frac{x-\mu}{\sigma})^2] \qquad (4.9)$$

A reference in statistics can be consulted for use of the χ^2 test to decide whether or not the sample data can actually be modeled well using the normal (or other) distribution. (For example, see the text by Hogg and Craig in the references.) It is easy to compute the mean and standard deviation from sample data and hence derive a normal distribution model. Many implementations will use the normal model because of its simplicity and because of other convenient mathematical properties, even when it is known to be a coarse approximation to the actual data.

Exercise 5

How could we estimate the following *a priori* probabilties? (a) that a customer in a market will buy spinach; (b) that a person at an ATM machine is an imposter; (c) that a just picked dark red cherry is bruised; (d) that a person over 40 years old has stomach cancer?

By using a *parametric model*, such as the normal distribution, to model a distribution of class samples, simple formulas are available for comparing probabilities for Bayesian decision-making as implemented in Figure 4.2. Once the distributions $p(x|\omega_i)$ are known for each class i, Figure 4.16 can be used to set the thresholds on values of x to separate the classes. Moreover, the probability model can be directly used to estimate the probability of error because there is now a formula for the error regions shown in Figure 4.15.

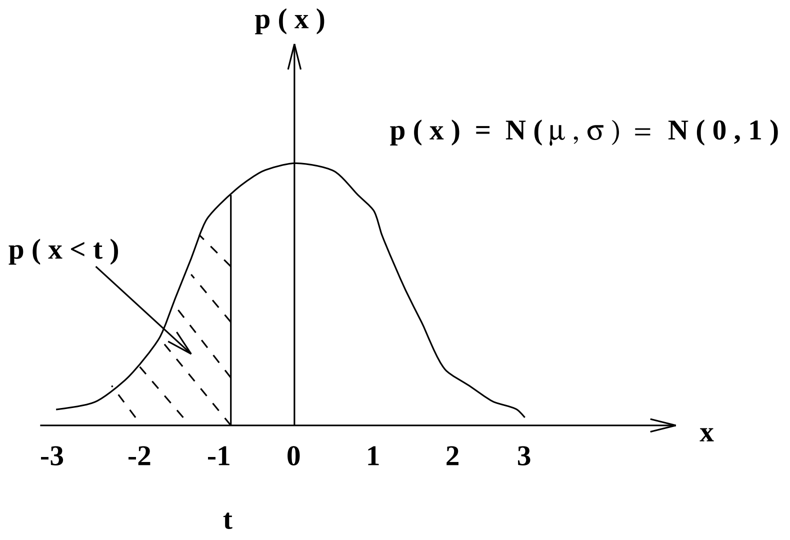
Exercise 6 classifying coins B

Refer to the previous exercise, which required measuring diameter and thickness of U.S. coins. Use the data for only the pennies, nickles, and dimes. (a) Let feature x be the thickness of the coin. Compute the mean and stardard deviation for each of the three classes. Are there thresholds t_1 and t_2 that would separate the classes and give an overall error rate of less than 5%? Explain. (b) Repeat (a) using x as the diameter of the coin.

4.11 Decisions using Multidimensional Data

In many real-world problems being tackled today, a dimension of $d = 10$ or more is common. As we saw previously, the Nearest Neighbor Classification procedure is defined for feature vectors of any dimension d. There are parametric probability models for multidimensional feature vectors $\mathbf{x}$; the reader should consult the references for their mathematical treatment. Here, we abstractly discuss the concept of multidimensional structure. A good intuitive grasp will reach a long way into the exciting current research cited in the references.

Consider two classes of samples in three dimensions, each shaped like a tree, with the two trees growing together. Class one data is shaped like a maple tree and is approximately a large sphere. Class two data is shaped like a pine, taller and much narrower than the maple, it is approximately an ellipsoid with major axis much larger than its two minor axes. Class one samples correspond to the leaves of the maple, while class two samples correspond



t	p(x<t)	t	p(x<t)	t	p(x<t)
-3.0	0.0014	-2.0	0.0227	-1.0	0.1587
-2.9	0.0019	-1.9	0.0287	-0.9	0.1841
-2.8	0.0026	-1.8	0.0359	-0.8	0.2119
-2.7	0.0035	-1.7	0.0446	-0.7	0.2420
-2.6	0.0047	-1.6	0.0548	-0.6	0.2743
-2.5	0.0062	-1.5	0.0668	-0.5	0.3085
-2.4	0.0082	-1.4	0.0808	-0.4	0.3446
-2.3	0.0107	-1.3	0.0968	-0.3	0.3821
-2.2	0.0139	-1.2	0.1151	-0.2	0.4207
-2.1	0.0179	-1.1	0.1357	-0.1	0.4602
		0.0	0.5000		

```
Use symmetry to extend the table values from 0.0 to 3.0. For example,
 p ( -2.0 < x < 1.0 ) = p ( -2.0 < x < 0.0 ) + p ( 0.0 < x < 1.0 )
  = [ p ( x < 0.0 ) - p ( x < -2.0 ) ] + p ( -1.0 < x < 0.0 )
    [ 0.5000 - 0.0227 ] + 0.1587 = 0.6360.
```

Figure 4.16: Normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$.

Exercise 7 On error rates due to thresholding

This exercise deals with the potential error in area computations due to the thresholding of an image in an attempt to separate an object from background. Make the following set of assumptions.

- The image of some object covers PRECISELY 3932 pixels of the 512x512 pixel image. (There are no mixed pixels; object boundaries correspond to pixel boundaries exactly. There is no blurring of neighboring pixels due to the lens.)
- Due to variations in its surface, the intensity of pixels of the image of the object is distributed as N(80,5) (this means normally distributed with mean 80 and standard deviation 5).
- Similarly, background intensity is distributed as N(50,10).
- The grey level of any single pixel is determined without regard to the grey level of neighboring pixels.

1. If the image is thresholded at intenisty 70 so that $LABEL[r,c] = 1\ if\ I[r,c] >= 70$ and LABEL[r,c] = 0 otherwise, what is the expected number of pixels labeled as OBJECT?
2. Where in the image are the pixels labeled BACKGROUND that truly should be OBJECT? (These are "false dismissals".)
3. Where in the image are the pixels labeled OBJECT that truly should be labeled BACKGROUND? (These are "false alarms".)
4. What is the percentage error expected in the computation of object area by merely counting the number of '1' pixels in the labeled image?
5. * Now suppose that *salt and pepper noise* is removed from the labeled image by creating a new image where any pixel is replaced by the values of the neighbors if all 4-neighbors have the other value. What is the percentage error expected in the computation of object area by merely counting the number of '1' pixels in this new labeled image?

to the needles of the pine. Moreover, suppose the pine tree grows through the crown of the maple and well above it. The problem of classifying an unknown 3D feature vector $\mathbf{x}$ requires relating it to the known sample structure in the 3D space. If $\mathbf{x}$ is within the crown of the maple, but not near the trunk of the pine, then $\mathbf{x}$ is more likely to be maple (class one). On the other hand, if $\mathbf{x}$ is outside the crown of the maple or close to the trunk of the pine, then $\mathbf{x}$ is more likely to be pine (class two). There are positions in the space that are ambiguous because the samples of the two classes overlap. The most important point is that an understanding of the structure of the samples in the d-dimensional space allows us not only to make informed decisions but also to understand our errors. The structure of the space can be represented by a large database of samples, data structures summarizing subsets of samples, or by parameterized geometric models of subsets of samples.

A second 3D example is also illuminating. Assume that the samples of class one are structured as a coil spring, or helix, and that the samples of class two are structured as a pencil, or rod, positioned as the axis of the helix. (Or, imagine two coil springs intertwined as they might be in a bin in a hardware store.) These classes are highly structured, in fact, are one-dimensional, and can be easily separated *once their structure is known.* A nearest-mean classifer is useless because the means are the same. Rescaling any of the dimensions is not going to work either because the samples will still be intertwined. Nearest neighbor classification will work, but we'll need to store a lot of samples. A practical alternative is to approximate the helical data by a union of many rods. A rod can be represented simply as a cylindrical section. Classification can be done by simple geometrical computations that check to see if unknown $\mathbf{x}$ is within any of the cylinders. A better alternative is to use a formula for the helix parameterized by its axis, radius, and rate of climb.

We note some important points in leaving these thought experiments. First, it is important to capture the *intrinsic structure and dimensionality* of the sample data. Structure can be represented by geometrical or statistical models: having models allows simple computations for our decisions as opposed to searching a large unstructured data base of samples. Secondly, the natural structure of the data may not be aligned with the axes of our measurement space. For example, the axis of the pine tree or the helix need not be along any of the axes $\mathbf{x}[1]$, $\mathbf{x}[2]$ or $\mathbf{x}[3]$. Methods for discovering structure or transforming coordinates are given in the references.

4.12 Machines that Learn

We pause to summarize the important point that the methods discussed in this chapter provide a basic type of machine learning called *supervised learning.* We have assumed that labeled samples were available for all of the classes that were to be distinguished; in other words, the teacher knew the structure of the data and the desired outcomes. *Unsupervised learning* or *clustering* can also be done; in unsupervised learning, the machine must also determine the class structure, that is, what the classes are and how many there are. The reader can consult the references to study this topic.

When nearest-neighbor classification is done, all the data samples are merely input into the memory and then accessed in order to recognize an unknown object. The recognition behavior of the machine is completely determined by the training data. When parametric models are used, the parameters of the class models are *learned* from the training data and then are used to model the entire space of possible objects. In the optional section below, supervised learning is achieved using discriminant functions that are designed to model the neurons of living organisms. Machine learning is currently an area of intense research and development, which the reader is encouraged to explore with additional reading.

4.13 * Artificial Neural Nets

Because of their learning capability, neurons of living organisms have been studied for application to machine learning. A simple model of a neuron is shown in Figure 4.17. Although the model is only an approximation to what is known from biology, it has become very important in its own right as a model for computation. Networks of such model neurons, called *artificial neural networks or ANNs*, have proved to be successful in many machine vision problems, especially because of their learning capability. ANNs can learn the complex structure of samples in multidimensional space using less memory than what is needed for nearest neighbor classification and can be implemented for massively parallel computation. Only a limited introduction to ANNs is given here; for more information about this large and still rapidly growing area, consult the references.

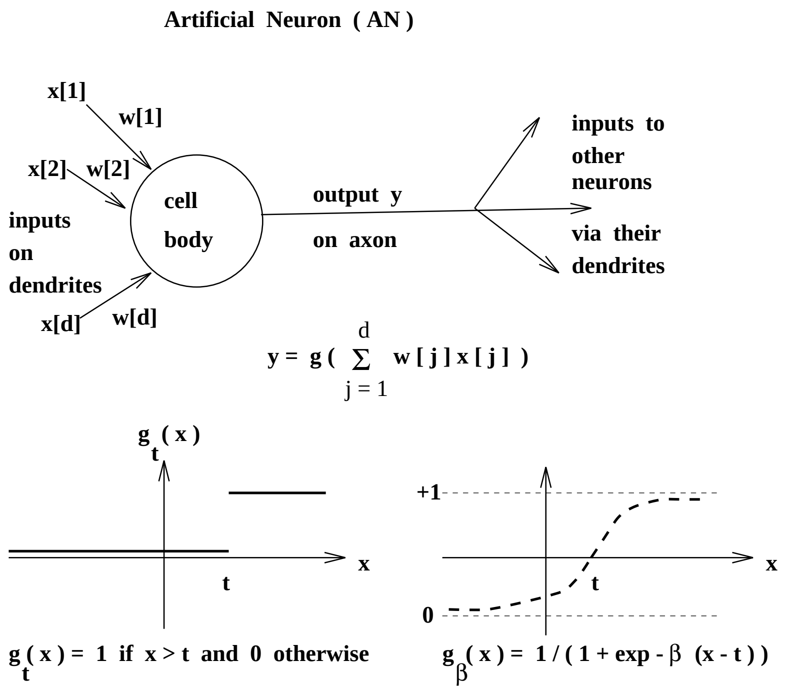


Figure 4.17: Simple model of a neuron and two possible output conditioning functions

The Perceptron Model

As Figure 4.17 shows, the neuron (AN) receives its d inputs $\mathbf{x}[j]$ via dendritic connections from other neurons, perhaps sensor cells. The cell body sums the inputs after multiplying each by a gain factor $\mathbf{w}[j]$. The neuron output y is sent along the single axon, which eventually branches into many dendritic connections, providing input to other neurons in the neural net. One model for determining y from the summed scaled inputs to the cell is to compare the sum with a threshold t and output $y = 1$ if the sum exceeds the threshold and output $y = 0$ otherwise. This binary output behavior is shown in the lower left of Figure 4.17. To get a smooth output behavior between 0 and 1, the sigmoid function, shown in the lower right of the figure, can be used. The parameter β is the slope or gain at $x = t$, which determines how the input value is scaled to compute the output value near $x = t$. For notational and programming convenience, the negative of the threshold t for the neuron is stored as $\mathbf{w}[0]$ and its corresponding input $\mathbf{x}[0]$ is set to 1.0 giving Equation 4.10. The neuron learns by adapting the weights $\mathbf{w}[j]$ to the input vectors $\mathbf{x}$ that it experiences.

$$y = g(\sum_{j=0,d} w[j]x[j]) \tag{4.10}$$

Exercise 8 simulating an AN

(a) Study the behavior of an AN with two inputs $\mathbf{x}[1]$ and $\mathbf{x}[2]$, with weights $\mathbf{w}[1] = 0.8$ and $\mathbf{w}[2] = 0.3$, threshold $t = 1.0$, and step function $g(x)$ at the output. Make a plot of the results by plotting a '1' when the output is 1 and a '0' when the output is 0 for the 16 possible input combinations where both $\mathbf{x}[1]$ and $\mathbf{x}[2]$ take values 0,1,2,3. (b) Create another plot, this time using the smooth sigmoid function with $\beta = 4$, keeping all other elements of the problem the same. Note that the outputs will now be real numbers rather than just 0 or 1.

Exercise 9 AND, OR, and NOT gates using an AN

1. Design a single AN that has the same behavior as an OR gate. Let $\mathbf{x}[1]$ and $\mathbf{x}[2]$ be the two inputs that can have Boolean values of 0 or 1 only. The output of the AN should be 1 when either or both of the inputs have value 1, and should be 0 when both inputs are 0. Recall that $\mathbf{x}[0] = 1$ and that the threshold is $-\mathbf{w}[0]$. Complete the set of weights to define the AN. Plot the four input combinations using 2D axes and show the decision boundary implemented by the AN.

2. Repeat the above question for an AND gate. The output of an AND gate is 1 only when both inputs are 1.

3. Show how a single input AN can behave as a NOT gate. If the input to the NOT gate is 0 then the output is 1, and if the input is 1 then the output is 0.

The computational capability of the simple artificial neuron is of great interest both in theory and practice. From Exercise 9 we learn that an AN can model AND, OR, and NOT gates. The significance of this is that *any* Boolean function can be implemented by cascading several ANs. From Exercise 10 we learn that a single AN cannot even implement the simple exclusive or function. Many other important functions cannot be implemented by a single AN: results published by Minski and Pappert (1987) discouraged research for a period. After a few years there was a spate of successful work with multilayer ANNs, which are more complex and not as limited in computational capability. We leave the theory of the computational capabilities of ANNs to further reading and return to look at a simple version of a training algorithm for a single AN.

Assume that two classes of 2D samples can be separated by a line: this would be the case for Figure 4.4 if we removed the single 'X' and 'O' from the overlap area. Clearly, one could take the parameters of the separating line and construct a neuron to make the classification decision. For 3D samples, we would use a separating plane; in d dimensions, we would need a hyperplane, but the concept and construction would be the same. Rather surprisingly, if a separating hyperplane exists for a two-class problem, then a simple learning algorithm exists to find the hyperplane formula from training samples from the two classes. The algorithm is given below. Proof that the algorithm converges to a separating hyperplane is beyond the scope of this text but can be found in Duda and Hart (73).

The perceptron learning algorithm begins with a random set of weights (including the threshold). It cycles through the labeled samples x and whenever the weight vector (perceptron) gives a positive output for a sample from Class 1, it subtracts gain $*$ x from the weight vector. Similarly, if there is a negative output for a sample from Class 2, then gain $*$ x is added to the weight vector. This policy moves the current separating line in the appropriate direction according to what was learned from the current sample. The gain controls the size of the change. The procedure training_pass carries out these adjustments. After all training samples are processed in one pass, the procedure check_samples is called to count how many samples are misclassified by an AN with these weights. If none are misclassified, then the algorithm exits with a solution. Otherwise, if the maximum allowed passes have not been done, then another training pass is made, this time with a halved gain factor. There are other implementations of the detailed control for the general algorithm.

Figure 4.18 shows the output from a program that implements the perceptron learning algorithm. By construction, all the Class 1 samples are below the line $y = 1 - x$, while all Class 2 samples are above that line. There is a corridor of space between these samples and the algorithm very quickly finds the line $-1 + 5/4x_1 + 5/4x_2 = 0$ to separate the classes. As the output shows, each sample from Class 1 produces a negative response while each sample from Class 2 produces a positive response.

Although the basic learning algorithm is simple, there are some nontrivial aspects. (1) What sequence of samples should be used to learn fast? Theory states that to guarantee convergence, each sample may have to be presented an arbitrary number of times. Some algorithms repeat training on a given sample until it is correctly classified and then move on to others. (2) Convergence will be affected by the gain factor used. The program used for the example output halved the gain factor with each pass through all the training samples. (3) For better performance on future samples, it may pay for the algorithm to search for a *best line* between the classes rather than just any line. (4) When training is taking a long time, how can we know whether or not it is due to the samples being inseparable? (5) How can we modify the learning algorithm so that in the case of samples that are linearly inseparable, we can find a line that yields the least misclassifications? These issues are left for possible outside research and experiments by the reader.

The Multilayer Feed-forward Network

A *feed-forward network* is a special type of ANN where each neuron in the network is

Compute weight vector **w** to discriminate Class 1 from 2.
S1 and **S2** are sets of n samples each.
gain is a scale factor used to change w when x is misclassified.
max_passes is maximum number of passes through all training samples.

```
procedure Perceptron_Learning(gain, max_passes, S1, S2)
{
input sample sets S1 and S2;
choose weight vector w randomly;
"let NE be the total number of samples misclassified"
NE = check_samples ( S1, S2, w);
while ( NE > 0 and passes < max_passes )
  {
    training_pass ( S1, S2, w, gain);
    NE = check_samples ( S1, S2, w);
    gain = 0.5 * gain;
    passes = passes + 1;
  }
report number of errors NE and weight vector w;
}
  procedure training_pass ( S1, S2, w, gain);
{
  for i from 1 to size of Sk
  {
    "scalar, or dot, product ∘ implements AN computation"
    take next x from S1;
    if ( $\mathbf{w} \circ \mathbf{x} > \mathbf{0}$ ) $w = w - gain * x$;
    take next x from S2;
    if ( $\mathbf{w} \circ \mathbf{x} < \mathbf{0}$ ) $w = w + gain * x$;
  }
}
```

Algorithm 3: A Perceptron Learning Algorithm for two linearly separable classes

Exercise 10 Perceptron to implement exclusive or (XOR)

Show that a single AN cannot make the exclusive OR decision by plotting the following input data and trying to find a separating line. The inputs giving a positive response are (0, 1), (1, 0) and the inputs giving a negative response are (0, 0), (1, 1).

Exercise 11 Program the perceptron learning algorithm

Write a program to implement the perceptron learning algorithm for arbitrary d-dimensional feature vectors **x**. Test it using 2D vectors and show that it can learn to discriminate as an OR gate and an AND gate. Show that learning does not converge for an XOR gate. Test on the following two classes of synthetic 3D samples: Class one is some set of random points in the first octant (x_1, x_2, x_3 all positive) while Class two is some set of points in any other octant.

```
Class 1 = { ( 0 , 0.5 ), ( 0.5 , 0 ), ( 0 , 0 ), ( 0.25 , 0.25 ) }
Class 2 = { ( 0 , 1.5 ), ( 1.5 , 0 ), ( 0.5 , 1 ), ( 1 , 0.5 ) }
Initial gain= 0.5
Limit to number of passes= 5
Number of samples in Class1= 4; Number of samples in Class2= 4

Training phase begins with weights:       -1     0.5     0.5

=====Adjust weights=====: gain= 0.5
 pattern vector x =        1       0     1.5
Input Weights:       -1     0.5     0.5
Output Weights:       -1     0.5    1.25

=====Adjust weights=====: gain= 0.5
 pattern vector x =        1     1.5       0
Input Weights:       -1     0.5    1.25
Output Weights:       -1    1.25    1.25

Weight Vector is:       -1    1.25    1.25  Classification for Class = 1

          Input Vector x / Response  / Error?

       1       0     0.5      -0.375    N
       1     0.5       0      -0.375    N
       1       0       0          -1    N
       1    0.25    0.25      -0.375    N

Weight Vector is:       -1    1.25    1.25  Classification for Class = 2

          Input Vector x / Response  / Error?

       1       0     1.5       0.875    N
       1     1.5       0       0.875    N
       1     0.5       1       0.875    N
       1       1     0.5       0.875    N

Errors for Class1: 0  Errors for Class2: 0
Final weights are:       -1    1.25    1.25
```

Figure 4.18: Output of computer perceptron learning program that learns the linear discriminant between two linearly separable classes.

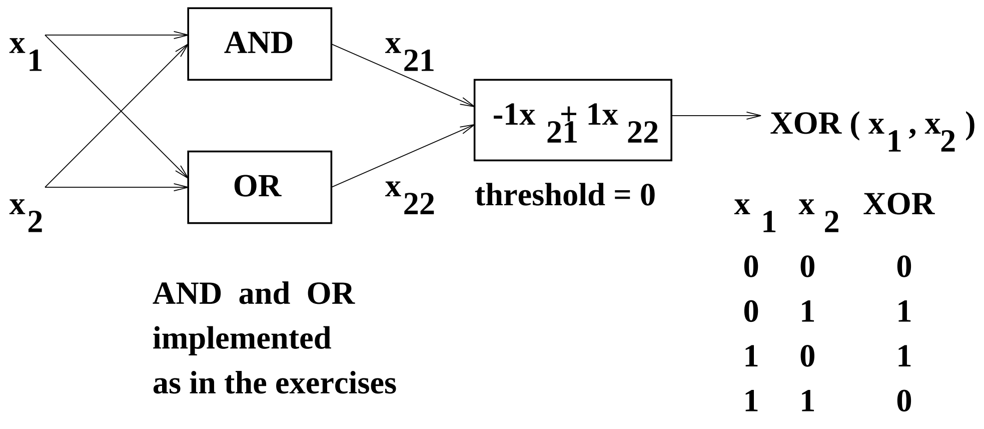


Figure 4.19: Implementation of XOR using a feedforward ANN

located at some level l. A neuron at level l receives input from all neurons of level $l-1$ and feeds its output to all neurons at level $l+1$. Refer to Figure 4.20. We can model the inputs to lowest level 1 neurons as sensor inputs and the outputs from highest level L neurons as classification results. The classification can be taken to be that c where output $\mathbf{y}[\mathbf{c}]$ is highest; or, all outputs can be considered to be a *fuzzy classification*. ANs between levels 1 and L are called *hidden units*. There is no feedback from any level to a lower level, thus the term "feed-forward". Because of this, the ANN works like a combinational circuit in the sense that its outputs are computed from its inputs without the use of memory about the prior sequence of inputs.

Previous exercises showed that single artificial neurons could have behavior equivalent to AND, OR, and NOT logic gates. This immediately implies that feed-forward layers of ANs can implement any combinational logic function. Thus, such networks are surprisingly powerful and can simulate the behavior of many different computer programs. Moreover, since ANs are not limited to Boolean values, they can represent very complex geometrical partitions of $d-$dimensional space and can adaptively learn such structure from training samples. Figure 4.19 shows how a feedforward ANN can compute the exclusive OR function, something that was impossible with a single AN. The first layer uses ANs to implement AND and OR as in the exercises. There is only one AN at the last level: it has weight vector $\mathbf{w} = [\mathbf{0}, \ -\mathbf{1}, \ \mathbf{1}]$ and outputs a 1 if and only if $-1x_1 + 1x_2$ is positive. In order to see how a multilayer ANN can capture the geometric structure of a complex set of samples, the reader should do the following exercises.

Exercise 12 an ANN for a triangular class structure in 2D

Construct a feed-forward ANN that yields output of 1 for all 2D points $\mathbf{x}$ inside the triangle with vertices (3,3), (6,6) and (9,1) and output of 0 for all 2D points outside of the triangle. Use the step function version of $g(x)$. Hint: use three ANs at the first level to establish the class boundaries that are the sides of the triangle and use one second level AN to integrate the three outputs from the first level.

Exercise 13 an ANN for a 3-class problem

Show how a 2 layer feed-forward network can recognize 2D input vectors from the following **disjoint** classes. Class 1 vectors are inside some triangle; class 2 vectors are inside some square, and class 3 vectors are inside some pentagon. Using the result of the above exercise, argue that an ANN exists to recognize each individual class versus its complement (you do not have to work with specific lines or equations, just call the subnetworks *triangle, square* and *pentagon*). The second layer output indicates the class of the input to the first layer.

A feed-forward network can learn by adapting its weights to a sequence of training samples during a learning phase. A learning algorithm, called the *back propagation algorithm* propagates classification errors from the output layers back toward the input layers. The sigmoid function is used to condition the output rather than thresholding in order to provide smooth control of the input/output relationship. Implementation and use of the back-propagation algorithm can be found in the references. Recently, the variety of successful applications of backpropagation and other learning algorithms has provided renewed excitement to the fields of both pattern recognition and machine learning. Consult the references to learn of other types of networks and their many applications.

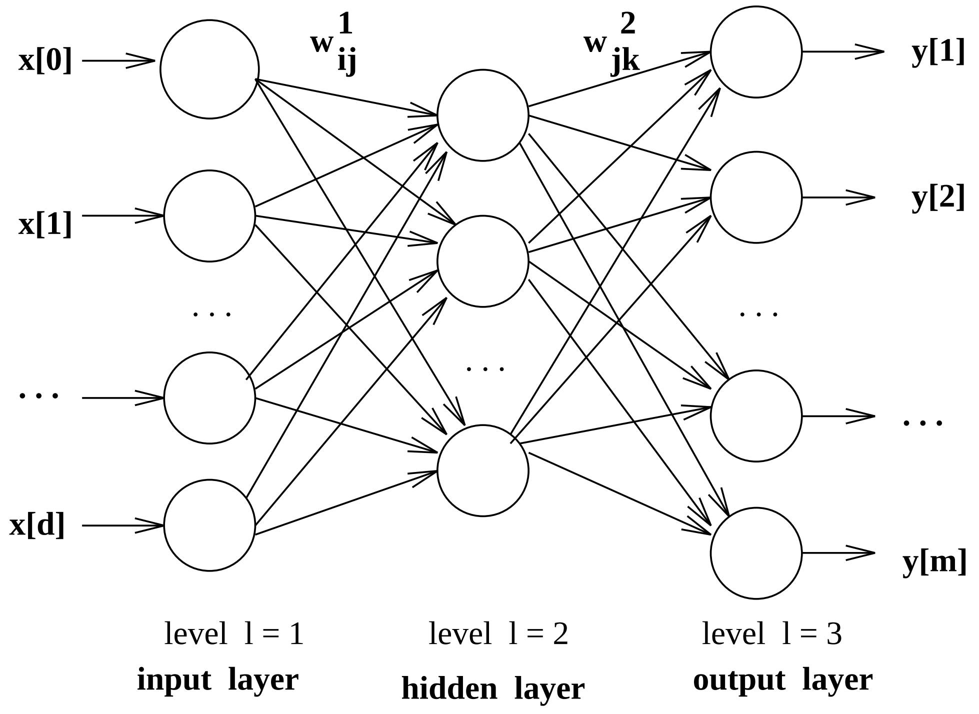


Figure 4.20: A multilayered feed-forward artificial neural network: all neurons in level l receive input from all neurons of level $l-1$ and feed output to all neurons of level $l+1$.

4.14 References

The early text by Duda and Hart (1973) is still a valuable reference for the problems and methods of classical statistical pattern recognition (there is a new edition). Another classical text is that of Fukunaga (1972,1990). These texts can be consulted for development of the Bayes classifier theory for d-dimensional feature vectors and they show the importance of the *covariance matrix* for modeling multidimensional structure. There are several good introductory texts on probability and statistics – one is by Hogg and Craig (1970) and another is by Feller (1957) – where one can study the theory of distributions such as the normal and chi-squared distributions. Jain *et al* (2000) is a survey of statistical pattern recognition containing many recent citations of contributions.

The *eigenvectors* of the covariance matrix give the natural directions of ellipsoidal clusters in space, while the corresponding *eigenvalues* give the spread of the samples. In addition, several ways of characterizing probability density are given. The recent texts by Schalkoff (1992) and Schurmann (1996) cover syntactic and structural pattern recognition in addition to statistical pattern recognition. A broad but brief treatment of ANNs can be found in the tutorial by Jain et al (1996); more extensive development is available in the texts by Haykin (1994), Hertz *et al* (1991) and by Schurmann (1996). The text by Tanimoto (1995) gives a good treatment of neural networks within the context of other learning schemes and also shows how to use symbolic features for input: implementations for both perceptron learning and back-propagation are given in LISP. For an exciting theoretical treatment on what perceptrons can and cannot compute, consult Minsky and Papert (1989) or the original edition (1969).

1. R. O. Duda and P. E. Hart (1973), **Pattern Classification and Scene Analysis**, John Wiley and Sons, New York.

2. W. Feller (1957), **An Introduction to Probability Theory and Its Applications; Vols I and II**, John Wiley & Sons.

3. S. Haykin (1994), **Neural Networks: a Comprehensive Foundation**, MacMillan College Publishing Co., New York.

4. J. Hertz, A. Krogh and R. Palmer (1991), **Introduction to the Theory of Neural Computation**, Addison-Wesley, Reading, Mass.

5. R. Hogg and A. Craig (1970), **Introduction to Mathematical Statistics**, The Macmillan Company.

6. A. K. Jain, J. Mao and K. M. Mohiuddin (1996), *Artificial Neural Networks: A Tutorial*, **IEEE Computer**, Vol. 29, No. 3, March 1996.

7. A. Jain, R. Duin and J. Mao (2000), *Statistical Pattern Recognition, A Review*, **IEEE-TPAMI**, Vol. 22, No. 1 (Jan 200)4-37.

8. K. Fukunaga (1990), **Introduction to Statistical Pattern Recognition, 2nd ed.**, Academic Press, New York.

9. A. Kulkarni (1994), **Artificial Neural Networks for Image Understanding**, Van Nostrand-Reinhold, New York.

10. M. Minsky and S. Papert (1989), **Perceptrons, 2nd ed.**, MIT Press, Cambridge, Mass.

11. J. G. Proakis, *Digital Communications*, McGraw-Hill, 1989.

12. J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, Vol 1, No. 1, 1986, pp 81-106.

13. R. Schalkoff (1992), **Pattern Recognition: statistical, structural, and neural approaches**, John Wiley and Sons, New York.

14. J. Schurmann (1996), **Pattern Classification: a Unified View of Statistical and Neural Approaches**, John Wiley and Sons, New York.

15. S. Tanimoto (1995), **The Elements of Artificial Intelligence with Common LISP, 2nd ed**, Computer Science Press, New York.