CHAPTER 19

VISUAL PATTERN RECOGNITION

WHAT WILL WE LEARN?

- What is visual pattern recognition and how does it relate to general pattern recognition?
- What are patterns and pattern classes?
- What is a pattern classifier?
- Which steps are normally needed to design, build, and test a visual pattern classifier?
- How can the performance of visual pattern classifiers be evaluated?

19.1 INTRODUCTION

This chapter presents the basic concepts of pattern recognition (also known as *pattern classification*) and introduces a few representative techniques used in computer vision. These techniques assume that an image has been acquired and processed and its contents have been represented using one or more of the techniques described in Chapter 18. The goal of pattern classification techniques is to assign a class to each image (or object within an image) based on a numerical representation of the image's (or object's) properties that is most suitable for the problem at hand.

Practical Image and Video Processing Using MATLAB®. By Oge Marques.
© 2011 John Wiley & Sons, Inc. Published 2011 by John Wiley & Sons, Inc.

Pattern classification techniques are usually classified into two main groups: statistical and structural (or syntactic). In this chapter, we exclusively focus on statistical pattern recognition techniques, which assume that each object or class can be represented as a *feature vector* and make decisions on which class to assign to a certain pattern based on distance calculations or probabilistic models. Since the techniques presented in this chapter work with numerical feature vectors, regardless of the meaning of their contents (and their relationship to the original images), they can also be applied to many other classes of problems outside image processing and computer vision. Nonetheless, since our focus is on *visual* pattern recognition, all examples will refer to images and objects within images, keeping this discussion consistent with the "big picture" presented in Chapter 1 and the feature extraction methods introduced in Chapter 18.

19.2 FUNDAMENTALS

In this section, we introduce some of the most important concepts and terminologies associated with pattern classification. The common goal of pattern classification techniques is to assign a *class* to an unknown *pattern* based on previously acquired knowledge about objects and the classes to which they belong. Figure 19.1 shows a schematic diagram indicating how a statistical pattern classifier processes numerical information from the feature vectors, computes a series of distances or probabilities, and uses those results to make decisions regarding which class label $C(\mathbf{x})$ should be assigned to each input pattern $\mathbf{x}$.

19.2.1 Design and Implementation of a Visual Pattern Classifier

The design and implementation of a visual pattern recognition system is usually an interactive process that also involves the selection and computation of features

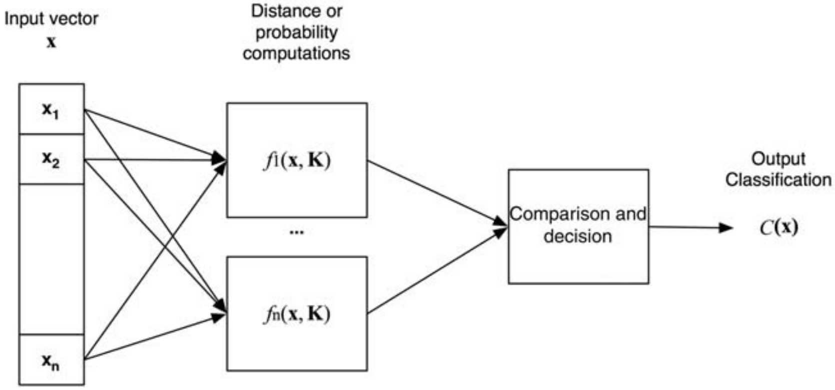

Input vector
x
Distance or probability computations
x_1
x_2
x_n
$f_1(\mathbf{x}, \mathbf{K})$
...
$f_n(\mathbf{x}, \mathbf{K})$
Comparison and decision
Output Classification
$C(\mathbf{x})$

FIGURE 19.1 Diagram of a statistical pattern classifier. Redrawn from [SS01].

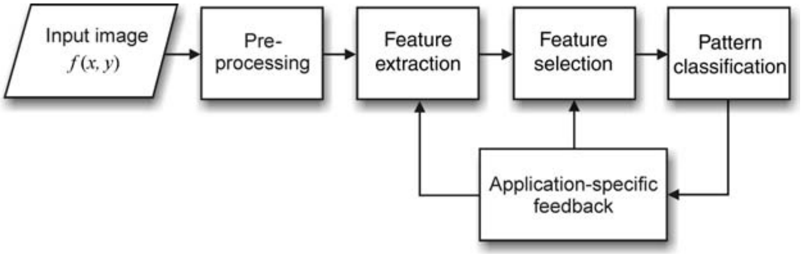


FIGURE 19.2 The interplay between feature extraction, feature selection, and pattern classification as a function of the application at hand. Adapted and redrawn from [Umb05].

from the images (or objects) that we want to classify. Moreover, as with most tasks in computer vision, the decisions are often application dependent. Figure 19.2 illustrates the process.

The design of a statistical visual pattern classifier usually consists of the following steps:

1. Define the problem and determine the number of classes involved.

   This is where it all begins. Legitimate questions to ask at this stage include the following: How many classes are there? How well do these classes describe the objects or images? Are certain classes subcategories of others? Is there a *reject class*[1] in this case?

2. Extract features that are most suitable to describe the images and allow the classifier to label them accordingly.

   This is the step where the interactive and application-dependent process of *preliminary* feature extraction and selection takes place. It is also the step where the designer of a machine vision solution is faced with many options of *types* of features (e.g., color based, texture based, boundary oriented, etc.) and the specific *methods* to extract them (e.g., color histograms, Tamura descriptors, Fourier descriptors, etc.). Refer to Chapter 18 for detailed information.

3. Select a classification method or algorithm.

   At this point, we can benefit from the vast array of tools and techniques in the field of data mining and machine learning and choose one that best suits our needs, based on their complexity, computational cost, training capabilities, and other properties. The selected technique can be as simple as a minimum distance classifier or as complex as a support vector machine (SVM).

4. Select a data set.

   At this stage, we collect representative images that can be used to train and test the solution. If the problem domain (e.g., object recognition) has associated

[1] A *reject class* is a generic class for objects that could not be successfully labeled as belonging to any of the other classes.

data sets that are publicly available and widely used (e.g., the Caltech 101 and Caltech 256 for object recognition), we should consider using them. Using standardized datasets allows for benchmarking of our solution against others.

5. Select a subset of images and use them to train the classifier.

   Many pattern classification strategies require a *training stage*, where a small subset of images is used to “teach” the classifier about the classes it should be able to recognize, as well as adjust some of the classifier’s parameters.

6. Test the classifier.

   At this step, we measure success and error rates, compute relevant figures of merit (e.g., precision and recall), and compile the main numerical results into representative plots (e.g., ROC curves).

7. Refine and improve the solution.

   After having analyzed the results computed in step 6, we might need to go back to an earlier step, process a few changes (e.g., select different features, modify a classifier’s parameters, collect additional images, etc.), and test the modified solution.

19.2.2 Patterns and Pattern Classes

A *pattern* can be defined as an arrangement of *descriptors* (or *features*). Patterns are usually encoded in the form of feature vectors, strings, or trees.[2]

A feature vector—as seen in Chapter 18—is an $n \times 1$ array of numbers corresponding to the descriptors (or features) used to represent a certain pattern:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}} \tag{19.1}$$

where n is the total number of features and T indicates the *transpose* operation.

The total number of features and their meaning will depend on the selected properties of the objects and the representation techniques used to describe them. This number can vary from one (if a certain property is discriminative enough to enable a classification decision) to several thousand (e.g., the number of points for a boundary encoded using Fourier descriptors, as seen in Section 18.4.3).

A *class* is a set of patterns that share some common properties. An ideal class is one in which its members are very similar to one another (i.e., the class has high intraclass similarity) and yet significantly different from members of other classes (i.e., interclass differences are significant). Pattern classes will be represented as follows: $\omega_1, \omega_2, \ldots, \omega_K$, where K is the total number of classes.

Figure 19.3 shows a simple example of a 2D plot of feature vectors representing the height and weight of a group of table tennis players and a group of sumo wrestlers:

$$\mathbf{x} = (x_1, x_2)^{\mathrm{T}} \tag{19.2}$$

where x_1 = weight and x_2 = height.

[2] Strings and trees are used in structural pattern classification methods and will not be discussed any further.

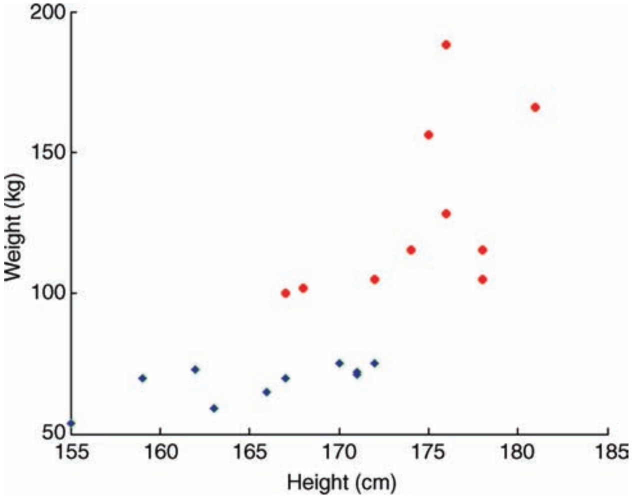


FIGURE 19.3 Example of two classes (*sumo wrestlers*—red circles—and *table tennis players*—blue diamonds) described by two measurements (*weight* and *height*).

In this example, ω_1 is the *sumo wrestlers* class, whereas ω_2 is the *table tennis players* class. A simple visual inspection allows us to see a clear separation between the two clusters of data points, which typically translates into a relatively easy task for a pattern classifier in charge of telling the two groups apart and classifying a new instance according to which group it most likely belongs to. Another observation that can be derived from the same figure is that the *weight* feature (x_1) is more discriminative than the *height* feature (x_2) in this particular problem.

19.2.3 Data Preprocessing

Before the numerical data (e.g., a collection of feature vectors) can be input to a pattern classifier, it is often necessary to perform an additional step known as *data preprocessing*. Common preprocessing techniques include the following:

- *Noise Removal* (also known as *Outlier Removal*): A preprocessing step where data samples that deviate too far from the average value for a class are removed, under the rationale that (a) there may have been a mistake while measuring (or extracting) that particular sample and (b) the sample is a poor example of the underlying structure of the class.
- *Normalization*: Feature vectors may need to be normalized before distance, similarity, and probability calculations take place. These are some representative normalization techniques [Umb05]:
  - *Unit Vector Normalization*: It enforces that all feature vectors have a magnitude of 1.

- *Standard Normal Density (SND)*: Here, each element of a feature vector **x** $(x_1, x_2, \ldots, x_n)$ is replaced by a normalized value $\tilde{x}_i$, given by

$$\tilde{x}_i = \frac{x_i - \mu}{\sigma} \tag{19.3}$$

where μ and σ are the mean and the standard deviation of the elements in **x**.

- *Other Linear and Nonlinear Techniques*: For example, *Min–Max Normalization* and *Softmax Scaling*: Their goal is to limit the feature values to a specific range, for example [0, 1].

- *Insertion of Missing Data*: In this (optional) last preprocessing step, additional data items—provided that they follow a similar probabilistic distribution and do not bias the results—are added to the data set.

19.2.4 Training and Test Sets

The process of development and testing of pattern classification algorithms usually requires that the data set be divided into two subgroups: the *training set*, used for algorithm development and fine-tuning, and the *test set*, used to evaluate the algorithm's performance. The training set contains a small (typically 20% or less) but representative subsample of the data set that can be selected manually or automatically (i.e., randomly). The size of the training set and the method used to build it are often dependent on the selected pattern classification technique. The goal of having two separate sets—one for designing, improving, and fine-tuning the algorithms, and the other for a systematic quantitative evaluation—is to avoid bias in reporting the success rates of the approach. After all, if the designer is allowed to work on the same data set the whole time, it is quite possible to tweak the solution enough to produce a nearly perfect performance on that particular collection of images. There would be no guarantee, however, that the same method and choice of parameters would work well for other images and data sets.

19.2.5 Confusion Matrix

The confusion matrix is a 2D array of size $K \times K$ (where K is the total number of classes) used to report raw results of classification experiments. The value in row i, column j indicates the number of times an object whose true class is i was labeled as belonging to class j. The main diagonal of the confusion matrix indicates the number of cases where the classifier was successful; a perfect classifier would show all off-diagonal elements equal to zero.

■ **EXAMPLE 19.1**

Figure 19.4 shows an example of a confusion matrix for four generic classes $\omega_1, \ldots,$ ω_4. A careful analysis of the confusion matrix shows that inputs labeled as class ω_3 were correctly classified all the time. Classification errors were highest (11%) for inputs labeled as class ω_2. The most common confusion incurred by the classifier was

	ω_1	ω_2	ω_3	ω_4
ω_1	97	0	2	1
ω_2	0	89	10	1
ω_3	0	0	100	0
ω_4	0	3	5	92

FIGURE 19.4 Example of 4 × 4 confusion matrix.

labeling an input of class ω_2 as class ω_3 (10% of the time). Moreover, the classifier's performance for class ω_1 is also worth commenting: although three inputs labeled as class ω_1 were incorrectly classified (two as class ω_3 and one as class ω_4), the classifier did not label any input from other classes as class ω_1 (i.e., the remaining values for the ω_1 column are all zeros).

19.2.6 System Errors

A quantitative performance analysis of pattern classifiers usually involves measuring *error rates*. Other measures, such as speed and computational complexity, may be important, of course, but error rates are essential. Error rates measure how many times a classifier incurred in a classification error, that is, classified an input object as class ω_p when the correct class is ω_q, $p \neq q$. The error rate of a classifier is usually determined empirically, that is, by measuring the number (percentage) of errors observed when testing the classifier against a test set.

■ **EXAMPLE 19.2**

Given the confusion matrix in Figure 19.4 and assuming that all classes had the same number of objects, the classifier's overall error rate would be

$$(3 + 11 + 0 + 8)/(4 \times 100) = 5.5\%.$$

19.2.7 Hit Rates, False Alarm Rates, and ROC Curves

Many visual pattern classification problems employ a two-class classifier. A classical computer vision example is the *object detection* task, where a computer vision algorithm is presented with an image and the question: "Is the object present in this image or not?" If the algorithm successfully answers *yes* (and points to where in the image the object is located) when the object is present, it is called a *true positive*. If the algorithm correctly answers *no* when the object is absent, it is called a *true negative*. There are two possible errors the algorithm can make: answering *yes* in the absence of an object (this is called a *false alarm* or *false positive*) or answering *no* when the object is present, that is, missing the object (this is called a *false negative*).

The cost of a false positive or a false negative is application dependent and can lead to quite different outcomes. In surveillance applications, for example, it will probably be best to have occasional false positives (e.g., alerting a human operator

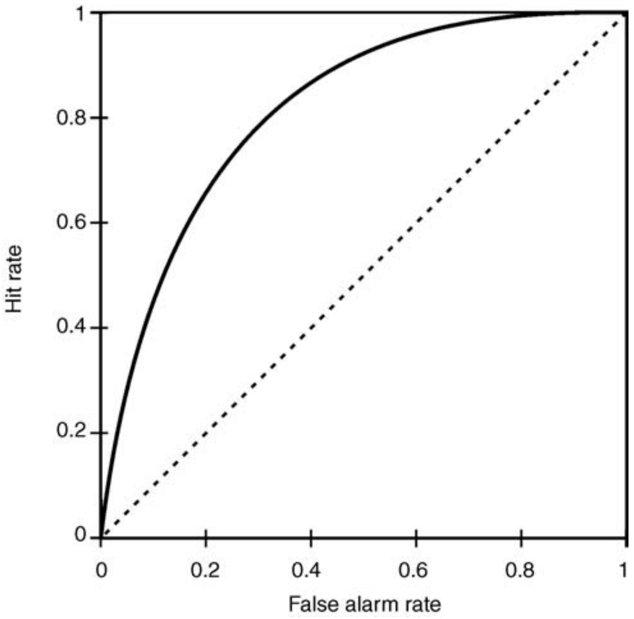


FIGURE 19.5 Example of ROC curve.

for the presence of a suspicious object on the screen where no such object exists) than miss real suspicious objects (or persons or events) altogether. In such case, a legitimate goal is to minimize the false negative rate, even if it can only be achieved by tolerating relatively high false positive rates.

The *receiver operating characteristic* (or simply *ROC*) curve is a plot that shows the relationship between the correct detection (true positive) rate (also known as *hit rate*) and the *false alarm* (false positive) *rate*. Figure 19.5 shows an example of generic ROC curve. It also shows a dashed straight line that corresponds to the performance of a classifier operating by chance (i.e., guessing *yes* or *no* at each time). The ideal ROC curve is one in which the “knee” of the curve is as close to the top left corner of the graph as possible, suggesting hit rate close to 100% with a false alarm rate close to zero.

19.2.8 Precision and Recall

Certain image processing applications, notably image retrieval, have the goal to retrieve *relevant* images while not retrieving *irrelevant* ones. The measures of performance used in image retrieval borrow from the field of (document) information retrieval and are based on two primary figures of merit: *precision* and *recall*. *Precision* is the number of relevant documents retrieved by the system divided by the total number of documents retrieved (i.e., true positives plus false alarms). *Recall* is the number of relevant documents retrieved by the system divided by the total number of relevant documents in the database (which should, therefore, have been retrieved).

Precision can be interpreted as a measure of exactness, whereas recall provides a measure of completeness. A perfect precision score of 1.0 means that every retrieved document (or image in our case) was relevant, but does not provide any insight as to whether all relevant documents were retrieved. A perfect recall score of 1.0 means that all relevant images were retrieved, but says nothing about how many irrelevant images might have also been retrieved.

Precision (P) and recall (R) measures can also be adapted to and used in classification tasks and expressed in terms of true positives (tp), false positives (fp), and false negatives (fn) as follows:

$$P = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}} \tag{19.4}$$

and

$$R = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}} \tag{19.5}$$

In this case, a precision score of 1.0 for a class ω_i means that every item labeled as belonging to class ω_i does indeed belong to class ω_i, but says nothing about the number of items from class ω_i that were not labeled correctly. A recall of 1.0 means that every item from class ω_i was labeled as belonging to class ω_i, but says nothing about how many other items were also incorrectly labeled as belonging to class ω_i.

■ EXAMPLE 19.3

Given the confusion matrix in Figure 19.4, the precision and recall per category can be calculated as follows:

$P_1 = 97/(97 + 0 + 0 + 0) = 100\%$
$P_2 = 89/(0 + 89 + 0 + 3) = 96.74\%$
$P_3 = 100/(2 + 10 + 100 + 5) = 85.47\%$
$P_4 = 92/(1 + 1 + 0 + 92) = 97.87\%$
$R_1 = 97/(97 + 0 + 2 + 1) = 97\%$
$R_2 = 89/(0 + 89 + 10 + 1) = 89\%$
$R_3 = 100/(0 + 0 + 100 + 0) = 100\%$
$R_4 = 92/(0 + 3 + 5 + 92) = 92\%$

In this case, the classifier shows perfect precision for class ω_1 and perfect recall for class ω_3.

Precision and recall are often interrelated and the cost of increasing one of them is an undesired decrease in the other. In the case of document (or image) retrieval systems, the choice of retrieving fewer documents boosts precision at the expense of a low recall, whereas the retrieval of too many documents improves recall at the expense of lower precision. This trade-off is often expressed in a plot, known as the *precision–recall* (or simply *PR*) *graph*. A PR graph is obtained by calculating the precision at various recall levels. The ideal PR graph shows perfect precision values at every recall level until the point where all relevant documents (and only those) have been retrieved; from this point on it falls monotonically until the point where recall reaches 1. Figure 19.6 shows an example of a generic PR graph.

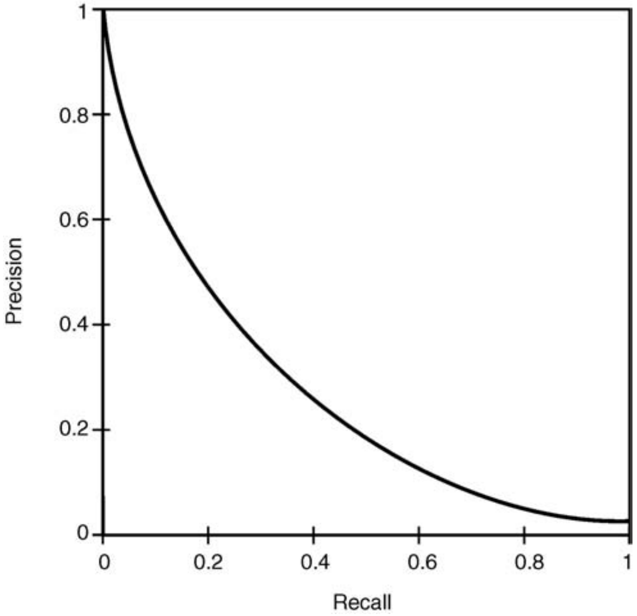


FIGURE 19.6 Example of precision–recall (PR) graph.

A more compact representation of the precision and recall properties of a system consists in combing those two values into a single figure of merit, such as the *F measure* (also known as *F1 measure*) that computes the weighted harmonic mean of precision and recall:

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{19.6}$$

■ EXAMPLE 19.4

An image retrieval system produced the following 10 ranked results for a search operation against a database of 500 images, of which 5 are relevant to the query:

Rank	Result
1	*R*
2	*R*
3	*N*
4	*R*
5	*N*
6	*N*
7	*N*
8	*R*
9	*N*
10	*R*

where *R* means *relevant* and *N* means *not relevant*.

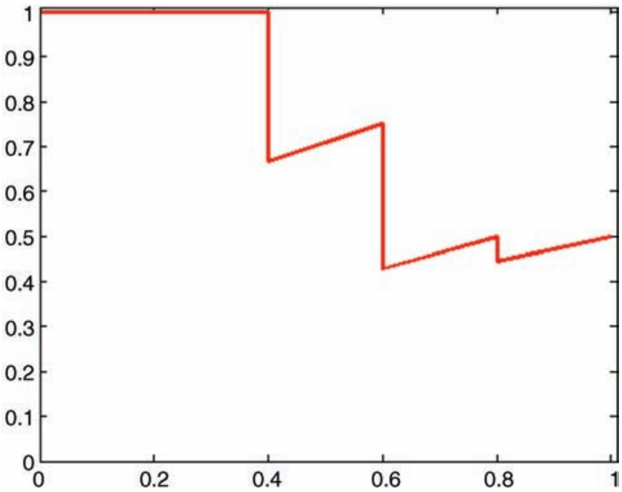


FIGURE 19.7 Precision–recall graph for Example 19.4.

If we calculate the precision at all recall levels, we will have the following results:

Recall	Precision
0.2	1.0000
0.4	1.0000
0.4	0.6667
0.6	0.7500
0.6	0.6000
0.6	0.5000
0.6	0.4286
0.8	0.5000
0.8	0.4444
1.0	0.5000

The corresponding PR graph is shown in Figure 19.7.

19.2.9 Distance and Similarity Measures

Two feature vectors can be compared with each other by calculating (i.e., measuring) the distance between them or, conversely, establishing their degree of similarity.

There are many distance measures in use in visual pattern classification. Given two feature vectors $\mathbf{a} = (a_1, a_2, \ldots, a_n)^{\mathrm{T}}$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)^{\mathrm{T}}$, the following are the equations for the most widely used distance measures:[3]

- Euclidean distance:

$$d_{\mathrm{E}} = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{19.7}$$

- Manhattan (or city block) distance:

$$d_{\mathrm{M}} = \sum_{i=1}^{n} |a_i - b_i| \tag{19.8}$$

- Minkowski distance:

$$d_{\mathrm{M}} = \left[\sum_{i=1}^{n} |a_i - b_i|^r \right]^{1/r} \tag{19.9}$$

  where r is a positive integer. Clearly, the Minkowski distances for $r = 1$ and $r = 2$ are the same as the Manhattan and Euclidean distances, respectively.

In MATLAB

Computing distances in MATLAB is straightforward, thanks to MATLAB's matrix handling abilities. For the Euclidean distance between two vectors `x` and `y`, we can use the `norm` function as an elegant alternative: `d_E = norm(x - y).`

Although distance measures are *inversely* related to the notion (and measure) of similarity, there are other similarity measures in the literature, such as:

- The vector inner product:

$$\sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{19.10}$$

- The Tanimoto metric, which establishes a percentage(%) of similarity and is expressed as

$$\frac{\sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n} a_i^2 + \sum_{i=1}^{n} b_i^2 - \sum_{i=1}^{n} a_i b_i} \tag{19.11}$$

[3] Some of these equations may seem familiar to you, since they appeared in a slightly different format back in Chapter 2. However, in Chapter 2 we were measuring distances between pixels, whereas in this chapter we are measuring distances between feature vectors in a feature space.

19.3 STATISTICAL PATTERN CLASSIFICATION TECHNIQUES

In this section, we present the basics of three statistical pattern classification techniques: the minimum distance classifier, the *k*-nearest neighbors (KNN) classifier, and the maximum likelihood (or Bayesian) classifier. The goal of this section is to present a few alternatives for building a visual pattern recognition using the knowledge from the previous chapters, particularly the feature extraction and representation techniques learned in Chapter 18. In Tutorial 19.1, you will have an opportunity for putting this knowledge into practice.

We have learned so far that objects' properties can be represented using *feature vectors* that are projected onto a *feature space*. If the features used to represent the objects (and the classes to which they belong) are properly chosen, the resulting points in the n-dimensional feature space will be distributed in a way that correlates proximity in the feature space with similarity among the actual objects. In other words, feature vectors associated with objects from the same class will appear close together as *clusters* in the feature space.

The job of a statistical pattern classification technique is to find a discrimination curve (or a *hypersurface*, in the case of an n-dimensional feature space) that can tell the clusters (and the classes to which they correspond) apart. Figure 19.8 illustrates the concept for a three-class classifier in a 2D feature space.

A statistical pattern classifier has n inputs (the features of the object to be classified, encoded into feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$) and one output (the class to which

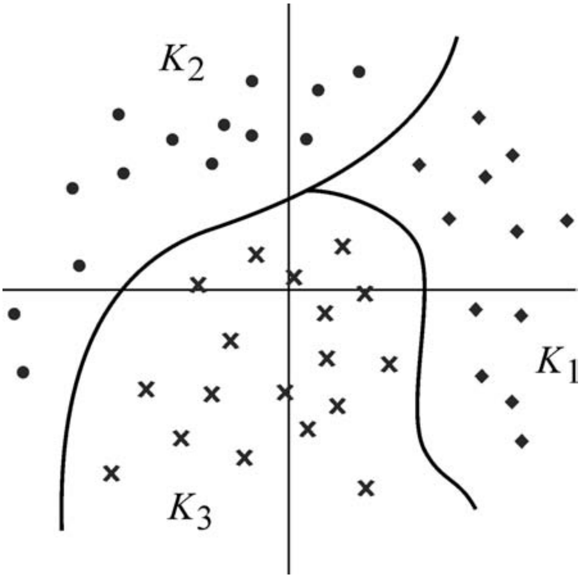


FIGURE 19.8 Discrimination functions for a three-class classifier in a 2D feature space.

the object belongs, $C(\mathbf{x})$, represented by one of the symbols $\omega_1, \omega_2, \ldots, \omega_W$, where W is the total number of classes). The symbols ω_w are called *class identifiers*.

Classifiers make comparisons among the representation of the unknown object and the known classes. These comparisons provide information to make decision about which class to assign to the unknown pattern. The decision of assigning an input pattern to class ω_i rather than another class ω_j is based on which side of the discrimination hypersurfaces among classes the unknown object sits on.

Mathematically, the job of the classifier is to apply a series of *decision rules* that divide the feature space into W disjoint subsets K_w, $w = 1, 2, \ldots, W$, each of which includes the feature vectors $\mathbf{x}$ for which $d(\mathbf{x}) = \omega_w$, where $d(\cdot)$ is the decision rule.

19.3.1 Minimum Distance Classifier

The minimum distance classifier (also known as the *nearest-class mean* classifier) works by computing a distance metric between an unknown feature vector and the centroids (i.e., mean vectors) of each class:

$$d_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\| \tag{19.12}$$

where d_j is a distance metric (between class j and the unknown feature vector $\mathbf{x}$) and $\mathbf{m}_j$ is the mean vector for class j, defined as

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x}_j \tag{19.13}$$

where N_j is the number of pattern vectors from class ω_j.

Figure 19.9 shows an example of two classes and their mean vectors.

The minimum distance classifier works well for few compact classes, but cannot handle more complex cases, such as the one depicted in Figure 19.10. In this case, there are two problems worth mentioning: (i) class A (clusters K_1 and K_4) is multimodal, that is, its samples lie in two disjoint (although they are compact) clusters, and the mean vector lies in a point in the feature space that is actually *outside* both clusters; (ii) classes B (cluster K_2) and C (cluster K_3) have quite irregular shapes that would potentially lead to different classification decisions for two unknown patterns situated at the same distance from their mean vectors. The latter problem can be alleviated by using a modified distance metric, known as *scaled Euclidean distance* (equation (19.14)), whereas the former problem usually calls for more complex classifier schemes.

$$\tilde{d}_{\mathrm{E}} = \|\mathbf{x} - \mathbf{x}_j\| = \sqrt{\sum_{i=1}^{n} [(x[i] - x_j[i])/\sigma_i]^2} \tag{19.14}$$

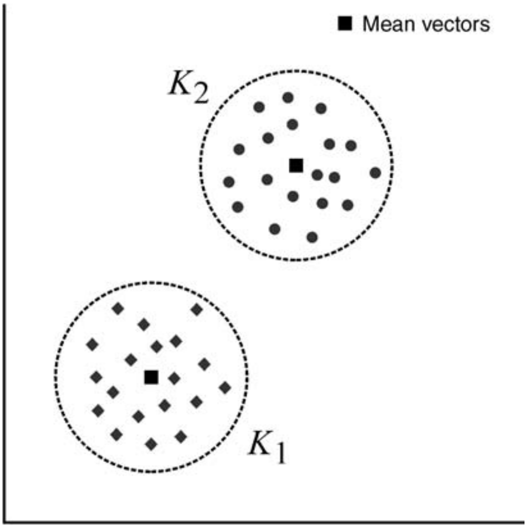


FIGURE 19.9 Example of two classes and their mean vectors.

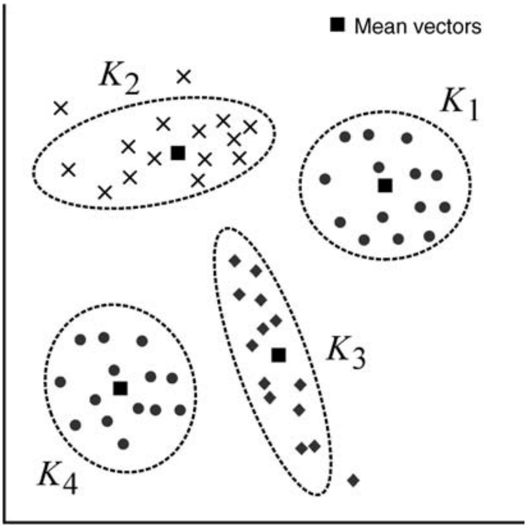


FIGURE 19.10 Example of three classes with relatively complex structure.

where $\tilde{d}_\text{E}$ is the scaled Euclidean distance between an unknown pattern and a class j, $\mathbf{x}$ is the feature vector of the unknown pattern, $\mathbf{x}_j$ is the mean vector of class j, σ_i is the standard deviation of class j along dimension i, and n is the dimensionality of the feature space.

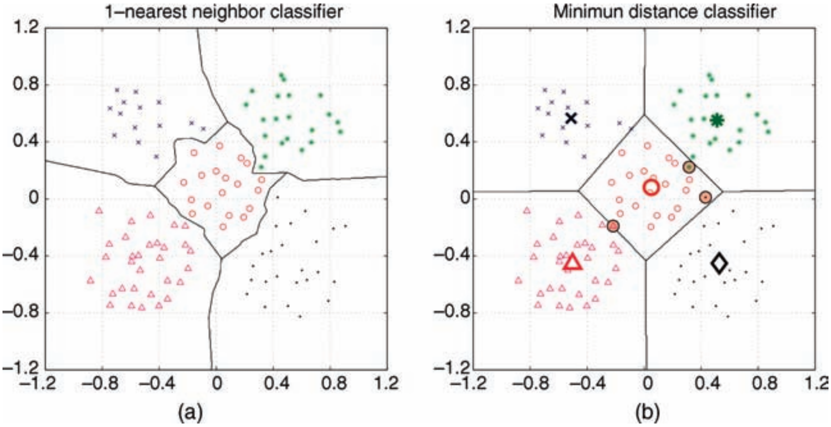


FIGURE 19.11 (a) Example of a KNN classifier ($k = 1$) for a five-class classifier in a 2D feature space (obtained using the STPRTool toolbox). (b) Minimum distance classifier results for the same data set.

19.3.2 *k*-Nearest Neighbors Classifier

A k-nearest neighbors (KNN) classifier works by computing the distance between an unknown pattern's feature vector $\mathbf{x}$ and the k closest *points*[4] to it in the feature space, and then assigning the unknown pattern to the class to which the majority of the k sampled points belong. The main advantages of this approach are its simplicity (e.g., no assumptions need to be made about the probability distributions of each class) and versatility (e.g., it handles overlapping classes or classes with complex structure well). Its main disadvantage is the computational cost involved in computing distances between the unknown sample and many (potentially *all*, if a brute force approach is used) stored points in the feature space.

Figure 19.11a illustrates the concept for a five-class classifier in a 2D feature space, where $k = 1$. It clearly shows that the KNN classifier is able to derive irregularly shaped discrimination functions among classes. This is in contrast to the minimum distance classifier, which would be constrained to using only straight lines as discrimination functions, as shown in Figure 19.11b, which also highlights the mean vectors for each class and the three data points that would be left out of their classes.

19.3.3 Bayesian Classifier

The rationale behind Bayesian classifiers is that a classification decision can be made based on the probability distributions of the training samples for each class; that is, an unknown object is assigned to the class to which it is *more likely* to belong based on the observed features.

[4] Note that we are referring to the k-nearest points, *not* the total number of classes, denoted as K.

The mathematical calculations performed by a Bayesian classifier require three probability distributions:

- The *a priori* (or *prior*) probability for each class ω_k, denoted by $P(\omega_k)$.
- The unconditional distribution of the feature vector representing the measured pattern **x**, denoted by $p(\mathbf{x})$.
- The class conditional distribution, that is, the probability of **x** given class ω_k, denoted by $p(\mathbf{x}|\omega_k)$.

These three distributions are then used, applying Bayes' rule, to compute the *a posteriori* probability that a pattern **x** comes from class ω_k, represented as $p(\omega_k|\mathbf{x})$, as follows:

$$p(\omega_k|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_k)P(\omega_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\omega_k)P(\omega_k)}{\sum_{k=1}^{W} p(\mathbf{x}|\omega_k)P(\omega_k)} \tag{19.15}$$

The design of a Bayes classifier requires that the prior probability for each class ($P(\omega_k)$) and the class conditional distribution ($p(\mathbf{x}|\omega_k)$) be known. The prior probability is easy to compute, based on the number of samples per class and the total number of samples. Estimating the class conditional distribution is a much harder problem, though, and it is often handled by modeling probability density function (PDF) of each class as a Gaussian (normal) distribution.

In MATLAB

There are many excellent MATLAB toolboxes for statistical pattern classification available on the Web (the "On the Web" section at the end of the chapter lists a few of them).

19.4 TUTORIAL 19.1: PATTERN CLASSIFICATION

Goal

The goal of this tutorial is to learn how to use MATLAB and publicly available toolboxes and data sets to build a visual pattern classifier.

Objectives

- Learn how to build a small optical character recognition (OCR) solution to classify digits (between 0 and 9) using `regionprops` and a KNN classifier.[5]
- Learn how to prepare and process the training and test data sets.

[5] The KNN classifier—and some of the supporting functions—used in this tutorial comes from the *Statistical Pattern Recognition Toolbox (STPRtool)*, available at http://cmp.felk.cvut.cz/cmp/software/stprtool/ .

- Learn how to perform feature selection.
- Learn how to present the classification results using a confusion matrix.

What You Will Need

- Scripts and test images available from the book web site (`ocr_example.zip`).

Procedure

1. Download the `ocr_example.zip` file and unzip its contents, keeping the folder structure. This file contain all the scripts and data you need, organized into meaningful folders.
2. Run script `load_data.m`. This script will load the 1000 training images and 1000 test images (10 images per character between 0 and 9) to the proper folders.

```
load_data
```

3. Examine the contents of script `load_data.m` to better understand *how* it does what it does.
4. Run script `preprocess_images.m`. This script will preprocess the training and test images and perform outlier removal.

```
preprocess_images
```

Question 1 What type of preprocessing operations are applied to all training and test images?

Question 2 How are outliers identified and handled?

5. Extract features from preprocessed training images using `regionprops` and selecting two properties with potential discriminative power: *eccentricity* and *Euler number*.
6. Store the results into a 2 × 1000 array corresponding to the 1000 feature vectors (of size 2 × 1 each).

```
fv = zeros(2,Ntrain);
for k = 1:Ntrain
    class = trn_data.y(k);
    I = imread([out_dir, sprintf('train_image_%02d_%02d_bw.png',...
        class,rem(k,100))]);
    [L, N] = bwlabel(I);
    % compute features and  build feature vector using 2 properties
    stats = regionprops(L,'all');
    fv(1,k) = stats.Eccentricity;
    fv(2,k) = stats.EulerNumber;
end
```

7. Save feature vector to a MAT file.

```
save([out_dir,'train_fv.mat'], 'fv');
```

8. Format feature vectors in a way that the KNN classifier will understand.

```
trn_data_binary.X = fv;
trn_data_binary.y = trn_data.y;
```

9. Create a KNN classifier and organize data for classification.

```
model = knnrule(trn_data_binary, 1);
```

10. Plot the 2D feature space and inspect it carefully. You should see a plot identical to Figure 19.12.

```
plot_feature_space
```

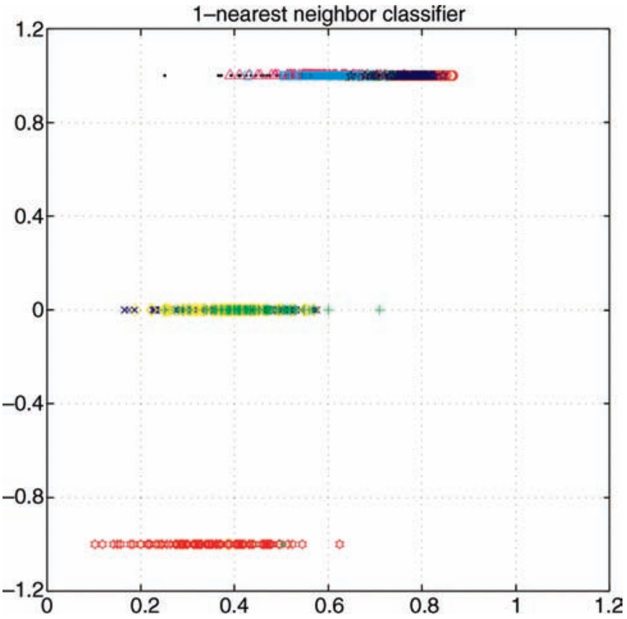


FIGURE 19.12 Feature space for training set. Obtained using the *Statistical Pattern Recognition Toolbox (STPRtool)*, available at http://cmp.felk.cvut.cz/cmp/software/stprtool/.

Question 3 Which feature is being represented on each axis of the plot?

Question 4 After inspecting the plot (and comparing it with that shown in Figure 19.11), what can you conclude so far?

11. Extract features from preprocessed test images using `regionprops` and selecting the same two properties used for the training set, namely, eccentricity and Euler number.
12. Store the results into a 2 × 1000 array corresponding to the 1000 feature vectors (of size 2 × 1 each).

```
fv_test = zeros(2,Ntest);
for k = 1:Ntest
    class = tst_data.y(k);
    I = imread([out_dir, sprintf('test_image_%02d_%02d_bw.png',...
        class,rem(k,100))]);
    [L, N] = bwlabel(I);
    stats = regionprops(L,'all');
    fv_test(1,k) = stats.Eccentricity;
    fv_test(2,k) = stats.EulerNumber;
end
```

13. Save feature vector to a MAT file.

```
save([out_dir,'test_fv.mat'], 'fv_test');
```

14. Format feature vectors in a way that the KNN classifier will understand.

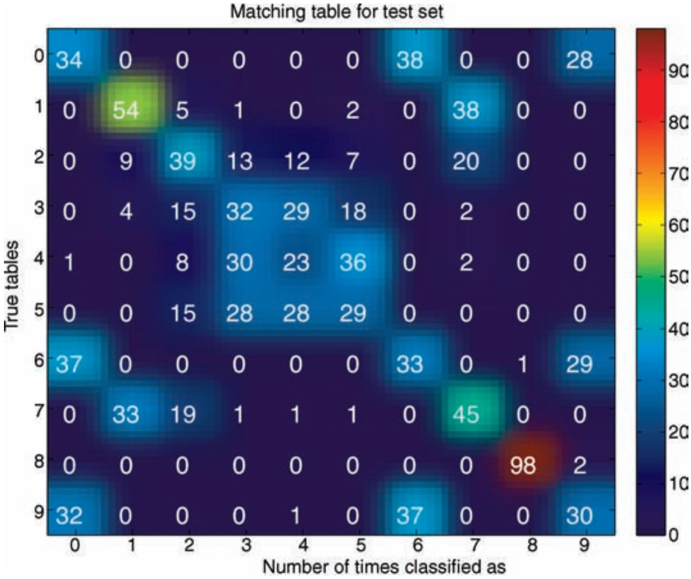
```
tst_data_binary.X = fv_test;
tst_data_binary.y = tst_data.y;
```

15. Run the KNN classifier to assign labels to the test images.

```
labels = knnclass(tst_data_binary.X,model);
display_kNN_results;
```

The results of running the kNN classifier on the test data set will show the following:

```
classification_result on test set data: 583 out of 1000 missclassified
class "0"  missclassified   66 times
class "1"  missclassified   46 times
class "2"  missclassified   61 times
class "3"  missclassified   68 times
class "4"  missclassified   77 times
class "5"  missclassified   71 times
class "6"  missclassified   67 times
```



	0	1	2	3	4	5	6	7	8	9
0	34	0	0	0	0	0	38	0	0	28
1	0	54	5	1	0	2	0	38	0	0
2	0	9	39	13	12	7	0	20	0	0
3	0	4	15	32	29	18	0	2	0	0
4	1	0	8	30	23	36	0	2	0	0
5	0	0	15	28	28	29	0	0	0	0
6	37	0	0	0	0	0	33	0	1	29
7	0	33	19	1	1	1	0	45	0	0
8	0	0	0	0	0	0	0	0	98	2
9	32	0	0	0	1	0	37	0	0	30

FIGURE 19.13 Confusion matrix with results of KNN classifier for the selected features. Obtained using the *Statistical Pattern Recognition Toolbox (STPRtool)*, available at http://cmp.felk.cvut.cz/cmp/software/stprtool/.

```
class "7"  missclassified   55 times
class "8"  missclassified    2 times
class "9"  missclassified   70 times
```

Figure 19.13 shows the resulting confusion matrix.

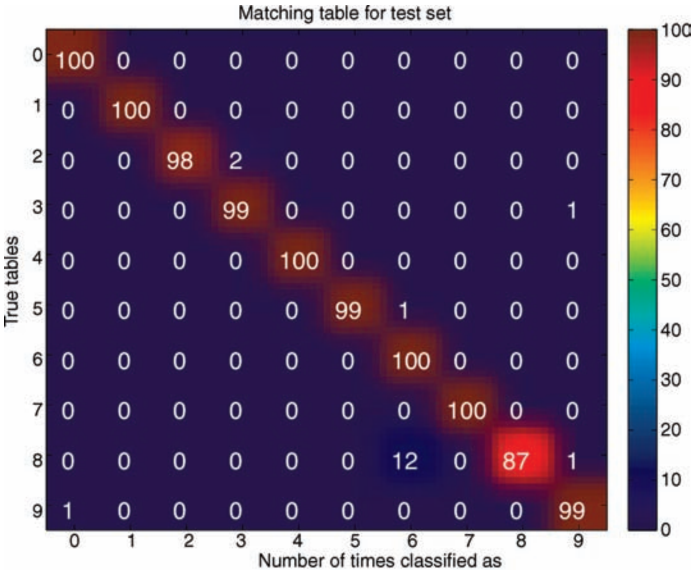
Question 5 Why are the results so poor (less than 42% success rate) overall?

Question 6 Why is the performance for one class (the digit 8) so much better than any other class?

Reflection

Our classifier did not perform as well as expected. It is time to look back at *each and every step* of our procedure and consider what needs to be changed or improved. At this point, all the evidence points in the direction of better, that is, more descriptive and discriminative, features.

In case you are wondering about the quality of the classifier, Figure 19.14 shows the resulting confusion matrix if we use the exact same classifier and test images, but different feature vectors. In this case, the actual gray values of the original 13 × 13 images were used as "features" that is, each image was represented using a 169 × 1 feature vector, without incurring in any of the preprocessing and feature extraction stages used in our solution. The results are remarkably better (98.2% overall success



	0	1	2	3	4	5	6	7	8	9
0	100	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0
2	0	0	98	2	0	0	0	0	0	0
3	0	0	0	99	0	0	0	0	0	1
4	0	0	0	0	100	0	0	0	0	0
5	0	0	0	0	0	99	1	0	0	0
6	0	0	0	0	0	0	100	0	0	0
7	0	0	0	0	0	0	0	100	0	0
8	0	0	0	0	0	0	12	0	87	1
9	1	0	0	0	0	0	0	0	0	99

FIGURE 19.14 Number of Confusion matrix with results of KNN classifier for the case where the images' gray values are used as "features." Obtained using the *Statistical Pattern Recognition Toolbox (STPRtool)*, available at http://cmp.felk.cvut.cz/cmp/software/stprtool/.

rate) than the ones we obtained with our choice of features (Figure 19.13). Another interesting remark: the worst-performing class (the number 8) for the modified design was the one that performed best in our original method.

Question 7 Does the alternative method scale well for larger (and more realistic) image sizes? Explain.

Question 8 Which changes would you make to our design (assuming the same dataset and classifier) and in which sequence would you experiment with them? Be precise.

WHAT HAVE WE LEARNED?

- Visual pattern recognition is the collection of methods used to recognize and classify patterns within images. It is a subset of the large research area of pattern recognition, in which the data used for the recognition and classification tasks correspond to the visual contents of a scene (and have been extracted from such scenes using one or more of the techniques described in Chapter 18).
- A *pattern* is an arrangement of *descriptors* (or *features*). The two most common arrangements are feature vectors (for quantitative descriptions) and strings (for

structural descriptions). A *pattern class* is a family of patterns that share a set of common properties.

- A *pattern classifier* is a (mathematical) method by which each sample in a data set is assigned a pattern class to which it is most likely to belong. Decision-theoretic pattern classifiers perform such assignment decisions based on mathematical properties such as distance measures between feature vectors or probability distributions of the pattern vectors within a class.

LEARN MORE ABOUT IT

- Pattern classification is a vast and complex field and entire books have been written on the topic. Two recent examples (both of which come with their own companion MATLAB toolbox) are [DHS01] and [vdHDdRT04].
- For information on structural pattern recognition techniques, including MATLAB code, refer to Chapter 8 of [DHS01] (and its companion manual) or Chapter 12 of [GWE04].

ON THE WEB

- Statistical Pattern Recognition Toolbox (STPRtool)
  http://cmp.felk.cvut.cz/cmp/software/stprtool/manual/
- PRTools: The Matlab Toolbox for Pattern Recognition
  http://prtools.org/
- DCPR (Data Clustering and Pattern Recognition) Toolbox
  http://neural.cs.nthu.edu.tw/jang/matlab/toolbox/DCPR/

19.5 PROBLEMS

19.1 Given the confusion matrix in Figure 19.15, use MATLAB to calculate the precision and recall per category.

19.2 Design and implement (in MATLAB) a machine vision solution to count the number of pennies, nickels, dimes, and quarters in an image containing U.S. coins.

	ω_1	ω_2	ω_3	ω_4
ω_1	97	0	2	1
ω_2	0	94	5	1
ω_3	0	0	100	0
ω_4	3	0	5	92

FIGURE 19.15 Confusion matrix for Problem 19.1.