PATTERN RECOGNITION

ROBI POLIKAR
Rowan University
Glassboro, New Jersey

1. INTRODUCTION

Pattern recognition stems from the need for automated machine recognition of objects, signals or images, or the need for automated decision-making based on a given set of parameters. Despite over half a century of productive research, pattern recognition continues to be an active area of research because of many unsolved fundamental theoretical problems as well as a rapidly increasing number of applications that can benefit from pattern recognition.

A fundamental challenge in automated recognition and decision-making is the fact that pattern recognition problems that appear to be simple for even a 5-year old may in fact be quite difficult when transferred to machine domain. Consider the problem of identifying the gender of a person by looking at a pictorial representation. Let us use the pictures in Fig. 1 for a simple demonstration, which include actual photographs as well as cartoon renderings. It is relatively straightforward for humans to effortlessly identify the genders of these people, but now consider the problem of having a machine making the same decision. What distinguishing features are there between these two classes—males and females—that the machine should look at to make an intelligent decision? It is not difficult to realize that many of the features that initially come to mind, such as hair length, height-to-weight ratio, body curvature, facial expressions, or facial bone structure—even when used in combination—may fail to provide correct male/female *classification* of these images. Although we can naturally and easily identify each person as male or female, it is not so easy to determine how we come to this conclusion, or more specifically, what features we use to solve this classification problem.

Of course, real-world pattern recognition problems are considerably more difficult then even the one illustrated above, and such problems span a very wide spectrum of applications, including speech recognition (e.g., automated voice-activated customer service), speaker identification, handwritten character recognition (such as the one used by the postal system to automatically read the addresses on envelopes), topographical remote sensing, identification of a system malfunction based on sensor data or loan/credit card application decision based on an individual's credit report data, among many others. More recently, a growing number of biomedical engineering-related applications have been added to this list, including DNA sequence identification, automated digital mammography analysis for early detection of breast cancer, automated electrocardiogram (ECG) or electroencephalogram (EEG) analysis for cardiovascular or neurological disorder diagnosis, and biometrics (personal identification based on biological data such as iris scan, fingerprint, etc.). The list of applications can be infinitely extended, but all of these applications share a common denominator: automated classification or decision making based on observed parameters, such as a signal, image, or in general a pattern, obtained by combining several observations or measurements.

This article provides an introductory background to pattern recognition and is organized as follows: The ter-

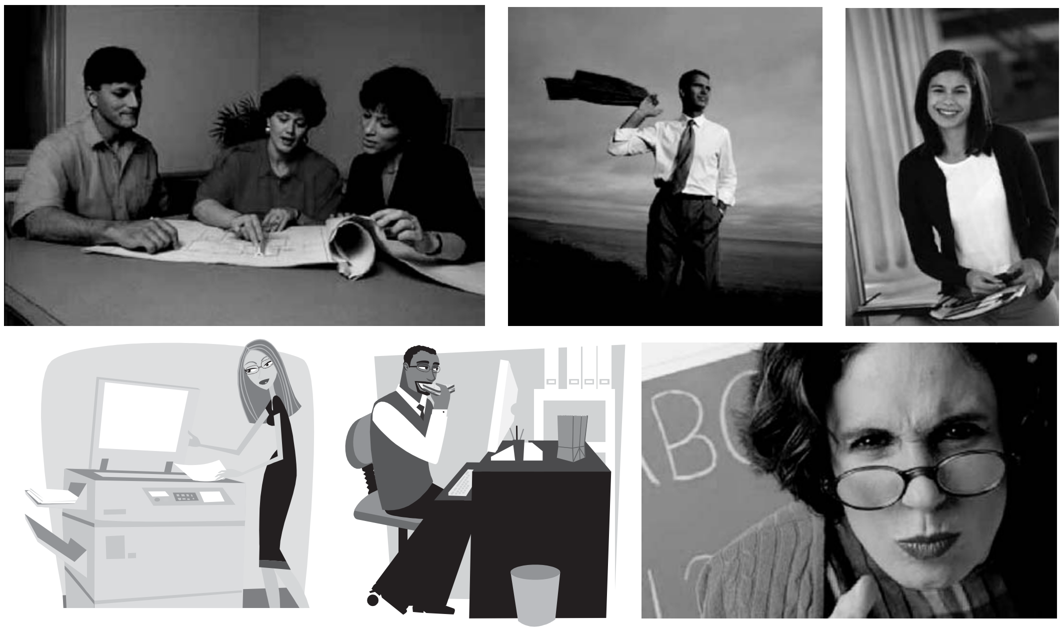


Figure 1. Pattern recognition problems that may be trivial for us may be quite challenging for automated systems. What distinguishing features can we use to identify above pictured people as males or females?

minology commonly used in pattern recognition is introduced first, followed by different components that make up a typical pattern recognition system. These components, which include data acquisition, feature extraction and selection, classification algorithm (model) selection and training, and evaluation of the system performance, are individually described. Particular emphasis is then given to different approaches that can be used for model selection and training, which constitutes the heart of a pattern recognition system. Some of the more advanced topics and current research issues are discussed last, such as kernel-based learning, combining classifiers and their many applications.

2. BACKGROUND AND TERMINOLOGY

2.1. Commonly Used Terminology in Pattern Recognition

A set of variables believed to carry discriminating and characterizing information about an object to be identified are called **features**, which are usually measurements or observations about the object. A collection of d such features, ordered in some meaningful way into d-dimensional column vector is the **feature vector**, denoted $\mathbf{x}$, which represents the signature of the object to be identified. The *d-dimensional* space in which the feature vector lies is referred to as the feature space. A d-dimensional vector in a d-dimensional space constitutes a point in that space. The category to which a given object belongs is called the *class* (or label) and is typically denoted by ω. A collection of features of an object under consideration, along with the correct class information for that object, is then called a *pattern*. Any given sample pattern of an object is also referred to as an *instance* or an *exemplar*. The goal of a pattern recognition system is therefore to estimate the correct label corresponding to a given feature vector based on some prior knowledge obtained through *training*. Training is the procedure by which the pattern recognition system learns the mapping relationship between feature vectors and their corresponding labels. This relationship forms the decision boundary in the d-dimensional feature space that separates patterns of different classes from each other. Therefore, we can equivalently state that the goal of a pattern recognition algorithm is to determine these *decision boundaries*, which are, in general, nonlinear functions. Consequently, pattern recognition can also be cast as a *function approximation* problem. Figure 2 illustrates these concepts on a hypothetical 2D, four class problem. For example, feature 1 may be systolic blood pressure measurement and feature 2 may be the weight of a patient, obtained from a cohort of elderly individuals over 60 years of age. Different classes may then indicate number of heart attacks suffered within the last 5-year period, such as none, one, two, or more than two.

The pattern recognition algorithm is usually trained using *training data*, for which the correct labels for each of the instances that makes up the data is *a priori* known. The performance of this algorithm is then evaluated on a separate *test* or *validation data*, typically collected at the same time or carved of the existing training data, for which the correct labels are also *a priori* known. Unknown

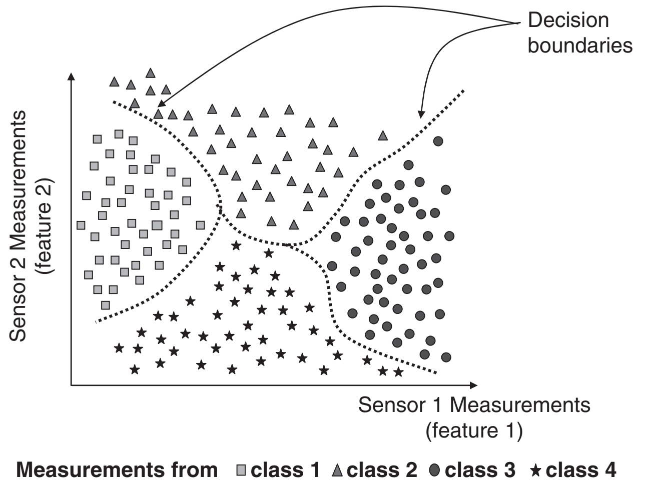


Figure 2. Graphical representation of data and decision boundaries.

data to be classified, for which the pattern recognition algorithm is trained, is then referred to as *field data*. The correct class labels for these data are obviously not known *a priori*, and it is the classifier's job to estimate the correct labels.

A quantitative measure that represents the cost of making a classification error is called the *cost function*. The pattern recognition algorithm is specifically trained to minimize this function. A typical cost function is the mean square error between numerically encoded values of actual and predicted labels. A pattern recognition system that adjusts its parameters to find the correct decision boundaries, through a learning algorithm using a training dataset, such that a cost function is minimized, is usually referred to as the *classifier* or more formally as the *model*. Incorrect labeling of the data by the classifier is an error and the cost of making a decision, in particular an incorrect one, is called the *cost of error*. We should quickly note that not all errors are equally costly. For example, consider the problem of estimating whether a patient is likely to experience myocardial infarction by analyzing a set of features obtained from the patient's recent ECG. Two possible types of error exist. The patient is in fact healthy but the classifier predicts that s/he is likely to have a myocardial infarction is known as a *false positive* (false alarm error), and typically referred to as *type I error*. The cost of making a type I error might be the side effects and the cost of administering certain drugs that are in fact not needed. Conversely, failing to recognize the warning signs of the ECG and declaring the patient as perfectly healthy is a *false negative*, also known as the *type II error*. The cost of making this type of error may include death. In this case, a type II error is costlier than a type I error. Pattern recognition algorithms can often be fine-tuned to minimize one type of error at the cost of increasing the other type.

Two parameters are often used in evaluating the performance of a trained system. The ability or the performance of the classifier in correctly identifying the classes of the training data, data that it has already seen, is called the *training performance*. The training performance is typically used to determine how long the training should continue, or how well the training data have been learned. The training performance is usually not a good indicator of the more meaningful *generalization performance*, which is the ability or the performance of the classifier in identifying the classes of previously unseen patterns.

In this article, we focus on the so-called *supervised* classification algorithms, where it is assumed that a training dataset with pre-assigned class labels is available. Applications exist where the user has access to data without correct class labels. Such applications are handled by *unsupervised* clustering algorithms. Unsupervised algorithms are not discussed in this article; however, some of the more commonly used clustering algorithms are mentioned in the last section of this article for users whose specific applications may call for unsupervised learning.

2.2. Common Issues in Pattern Recognition

The two-feature, four class hypothetical example shown in Fig. 2 represents a rather overly optimistic, if not an ideal scenario: Patterns from any given class are perfectly separable from those of other classes through some boundary, albeit a nonlinear one. In fact, such problems are often considered as *easy* for most pattern recognition applications. In most applications of practical interest, the data are not as cooperative. A more realistic scenario is the one shown in Fig. 3, where the patterns from different classes overlap in the feature space. Looking at data distribution in Fig. 3, one may think that it is impossible to draw a decision boundary that perfectly separates instances of one class from others, and that the algorithm's task is an impossible one. Not so. The task of the pattern recognition algorithms is to determine the decision boundary that provides the best possible generalization performance (on unseen data), and not one that provides perfect training performance. In fact, even if it were possible to find such a decision boundary that provides perfect separation of training data, such boundaries are usually not desired because much of the overlap is typically caused by noisy data, and finding a decision boundary that provides perfect training data classification would amount to learning the noise in the data.

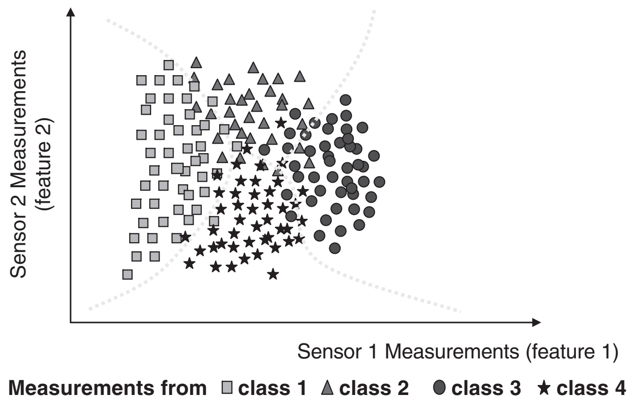


Figure 3. Uncooperative data are common in practical pattern recognition applications.

Learning noise invariably causes an inferior generalization performance on the test data. This phenomenon is observed often in pattern recognition and is referred to as *overtraining* or *overfitting*. Several approaches have been proposed to prevent overfitting, such as early termination of the training algorithm, or using a separate validation dataset, and adjusting the algorithm parameters until the performance on this validation dataset is optimized.

Uncooperative data can be because of a variety of external problems; for example, the dataset may be uncooperative because of the number of features being inadequate. For the hypothetical case mentioned above, if we were to add patient's age and cholesterol levels, the data (then in a 4D space) may be better separable. The opposite problem, presence of irrelevant features, can also be a problem. Another culprit for overlapping data is the outliers in the data. Even if the features are selected appropriately, outliers may always cause overlapping patterns (for example, it is not unusual to see people with high weight and high blood pressure and no apparent cardiovascular problems). It is therefore essential to use appropriate feature extraction/selection and outlier detection approaches to minimize the effects of uncooperative data.

Finally, a very important issue is the evaluation of classifier's performance. It is mentioned above that a test dataset is used for this purpose, but considering that the test dataset is really a subset of the original data, a portion of which were carved out for training, how can we ensure that the evaluation we conduct on this data represents the true performance of the system on never-before-seen field data?

All these issues lie within the domain of pattern recognition, and they are discussed below as we describe the individual components of a complete pattern recognition system.

3. COMPONENTS OF A PATTERN RECOGNITION SYSTEM

A classifier model and its associated training algorithm are all that are usually associated with pattern recognition. However, a complete pattern recognition system consists of several components, shown in Fig. 4, of which selection and training of such a model is just one component. We describe each component prior to actual model selection in this section, giving particular emphasis to model selection, training, and evaluation in subsequent sections.

3.1. Data Acquisition

Apart from employing an appropriate classification algorithm, one of the most important requirements for designing a successful pattern recognition system is to have adequate and representative training and test datasets. *Adequacy* ensures that a sufficient amount of data exists to learn the decision boundary as a functional mapping between the feature vectors and the correct class labels. There is no rule that specifies how much data is sufficient;

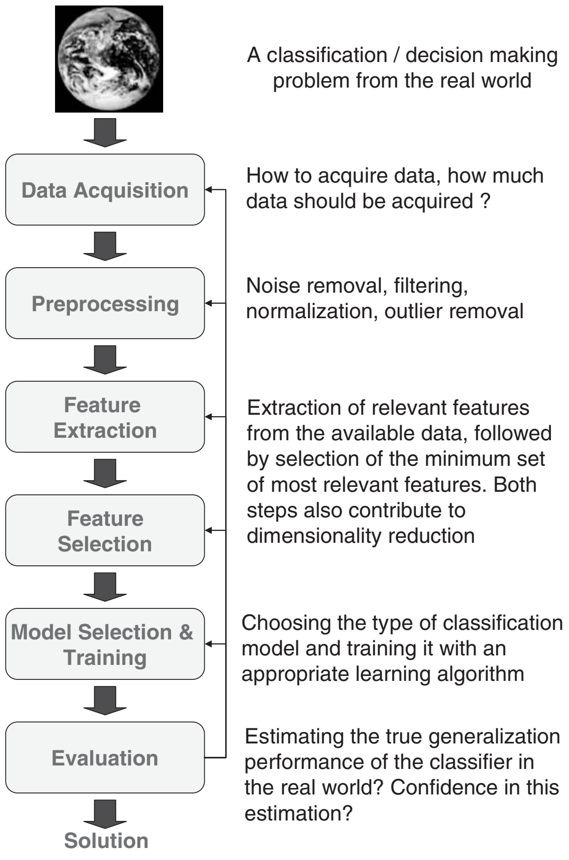

A classification / decision making problem from the real world
Data Acquisition
How to acquire data, how much data should be acquired ?
Preprocessing
Noise removal, filtering, normalization, outlier removal
Feature Extraction
Extraction of relevant features from the available data, followed by selection of the minimum set of most relevant features. Both steps also contribute to dimensionality reduction
Feature Selection
Model Selection & Training
Choosing the type of classification model and training it with an appropriate learning algorithm
Evaluation
Estimating the true generalization performance of the classifier in the real world? Confidence in this estimation?
Solution

Figure 4. Components of a pattern recognition system.

however, a general guideline indicates that there should either be at least 10 times the number of training data instances as there are adjustable parameters in the classifier (1), or 10 times the instances per class per feature, to reduce or minimize overfitting (2). *Representative data*, on the other hand, ensures that all meaningful variations of field data instances that the system is likely to see are sampled by the training and test data. This condition is often more difficult to satisfy, as it is usually not practical to determine whether the training data distribution adequately spans the entire space on which the problem is defined. An educated guess is usually all that is available to the designer in choosing the type of sensors or measurement schemes that will provide the data.

3.2. Preprocessing

An essential, yet often overlooked step in the design process is preprocessing, where the goal is to condition the acquired data such that noise from various sources are removed to the extend that it is possible. Various filtering techniques can be employed if the user has prior knowledge regarding the spectrum of the noise. For example, if the measurement environment is known to introduce high (low)-frequency noise, an appropriate low (high)-pass DIGITAL FILTER may be employed to remove the noise. ADAPTIVE FILTERS can also be used if the spectral characteristics of the noise are known to change in time.

Conditioning may also include normalization, as classifiers are known to perform better with feature values that lie in a relatively smaller range. Normalization can be done with respect to the mean and variance of the feature values or with respect to the amplitude of the data. In the former, instances are normalized such that the normalized data have zero mean and unit variance

$$x_i' = \frac{x_i - \mu_i}{\sigma_i}, \tag{1}$$

where x_i indicates the ith feature of original data instance, x_i' is its normalized value, μ_i is the mean, and σ_i is the standard deviation of x_i. In the latter, instances are simply divided by a constant so that all feature values are restricted to $[-1\ 1]$ range:

$$\mathbf{x} = \frac{\mathbf{x}}{\sqrt{\sum_{i=1}^{d} (x_i)^2}}. \tag{2}$$

Finally, if great magnitude differences exist between the individual feature values, a logarithmic transformation, for example, can be used to reduce the dynamic range of the feature values.

Preprocessing should also include outlier removal when possible. For low-dimensional problems ($d \leq 3$), the data can be plotted, which often provides visual clues on whether any outliers are present. For data with $d > 3$, Mahalanobis distance can be used, particularly if the data follow a Gaussian or near-Gaussian distribution. In this case, one computes the Mahalanobis distance of each data point from the distribution of training data instances for the class it belongs, and if this distance is larger than a threshold (such as some multiple of the standard deviation of the data), that instance can be considered as an outlier. The Mahalanobis distance of instance $\mathbf{x}$ from the training data of instances of class ω_j can be computed as

$$M_j = (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j), \tag{3}$$

where $\boldsymbol{\mu}_j$ is the mean of class ω_j instances and Σ_j is their covariance matrix.

3.3. Feature Extraction

Both feature extraction and feature selection steps (discussed next) are in effect dimensionality reduction procedures. In short, the goal of *feature extraction* is to find preferably small number of features that are particularly distinguishing or informative for the classification process, and that are invariant to irrelevant transformations of the data. Consider the identification of a cancerous tissue from an MRI image: The shape, color contrast ratio of this tissue to that of surrounding tissue, 2D Fourier spectrum, and so on are all likely to be relevant and distinguishing features, but the height or eye color of the patient are probably not. Furthermore, although the shape is a relevant feature, tumors whose shapes are small or large, that are a few centimeters to the left or right, or that are rotated in one direction or another, are still tumors. Therefore, an appropriate transformation may be necessary to obtain a translation, rotation, and scale invariant version of the shape feature. The goal of *feature selection*, on the other hand, is to select a yet smaller subset of the extracted features that are deemed to be the *most* distinguishing and informative ones.

The importance of dimensionality reduction in pattern recognition cannot be overstated. A small but informative set of features significantly reduces the complexity of the classification algorithm, the time and memory requirements to run this algorithm, as well as the possibility of overfitting. In fact, the detrimental effects of a large number of features are well known within the pattern recognition community, and affectionately referred to as the *curse of dimensionality.* It is therefore important to keep the number of features as few as possible, while ensuring that enough discriminating power is retained. The Ockham's Razor, the well-known philosophical argument of William of Ockham (1284–1347), that *entities are not to be multiplied without necessity* is often mentioned when the virtues of dimensionality reduction are discussed.

Feature extraction is usually obtained from a mathematical transformation on the data. Some of the most widely used transformations are linear transformations, such as PRINCIPAL COMPONENT ANALYSIS and *linear discriminant analysis*.

3.3.1. Principal Component Analysis (PCA). PCA, also known as Karhunen–Loève transformation, is one of the oldest techniques in multivariate analysis and is the most commonly used dimensionality reduction technique in pattern recognition. It was originally developed by Pearson in 1901 and generalized by Loéve in 1963.

The underlying idea is to project the data to a lower dimensional space, where most of the information is retained. In PCA, it is assumed that the information is carried in the variance of the features, that is, the higher the variation in a feature, the more information that feature carries. Hence, PCA employs a linear transformation that is based on preserving the most variance in the data using the least number of dimensions. The data instances are projected onto a lower dimensional space where the new features best represent the entire data in the *least squares sense*. It can be shown that the optimal approximation, in the least square error sense, of a d-dimensional random vector $\mathbf{x} \in \Re^d$ by a linear combination of $d' < d$ independent vectors is obtained by projecting the vector $\mathbf{x}$ onto the eigenvectors $\mathbf{e_i}$ corresponding to the largest eigenvalues λ_i of the covariance matrix (or the scatter matrix) of the data from which $\mathbf{x}$ is drawn. The eigenvectors of the covariance matrix of the data are referred to as *principal* axes of the data, and the projection of the data instances on to these principal axes are called the *principal components*. Dimensionality reduction is then obtained by only retaining those axes (dimensions) that account for most of the variance, and discarding all others.

Figure 5 illustrates PCA, where the principal axes are aligned with the most variation in the data. In this 2D case, *Principal Axis 1* accounts for more of the variation,

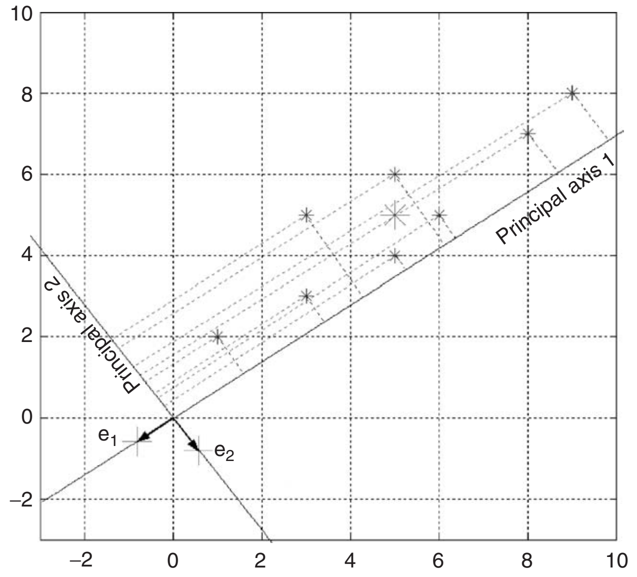


Figure 5. Principal axes are along the eigenvectors of the covariance matrix of the data.

and dimensionality reduction can be obtained by projecting the 2D original data onto the first principal axis, thereby obtaining 1D data that best represent the original data.

In summary, PCA is equivalent to walking around the data to see from which angle one gets the best view. The rotation of the axes is done in such a way that the new axes are aligned with the directions of maximum variance, which are the directions of eigenvectors of the covariance matrix. Implementation details can be found in the article on PRINCIPAL COMPONENT ANALYSIS.

One minor problem exists with PCA, however: Although it provides the best representation of the data in a lower dimensional space, no guarantee exists that the classes in the transformed data are better separated than they are in the original data, which is because PCA does not take class information into consideration. Ensuring that the classes are best separated in the transformed space is better handled by the linear discriminant analysis (LDA).

3.3.2. Linear Discriminant Analysis (LDA). The goal of LDA (or *Fisher Linear Discriminant*) is to find a transformation such that the *intercluster distances* between the classes are maximized and *intracluster distances* within a given class are minimized in the transformed lower dimensional space. These distances are measured using *between* and *within scatter matrix*, respectively, as described below.

Consider a multiclass classification problem and let C be the number of classes. For the ith class, let $\{\mathbf{x}_i\}$ be the set of patterns in this class, $\mathbf{m_i}$ be the mean of vectors $\mathbf{x} \in \{\mathbf{x}_i\}$, and n_i be the number of patterns in $\{\mathbf{x}_i\}$. Let $\mathbf{m}$ be the mean of all patterns in all C classes. Then, the within scatter matrix $\mathbf{S_W}$ and between scatter matrix $\mathbf{S_B}$ are de-

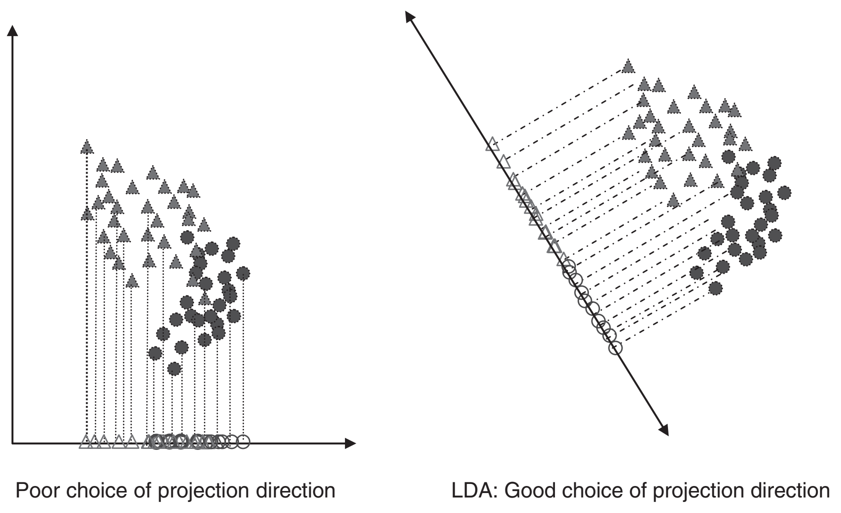

Poor choice of projection direction
LDA: Good choice of projection direction

Figure 6. Linear discriminant analysis.

fined as follows (1):

$$\mathbf{S}_W = \sum_{i=1}^{C} \sum_{\mathbf{x} \in X_i} (\mathbf{x} - \mathbf{m}_i) \cdot (\mathbf{x} - \mathbf{m}_i)^T$$
$$\mathbf{S}_B = \sum_{i=1}^{C} n_i (\mathbf{m} - \mathbf{m}_i) \cdot (\mathbf{m} - \mathbf{m}_i)^T, \tag{4}$$

where T denotes the transpose operator. The transformation, which is also the projection from the original feature space onto a lower dimensional feature space, can be expressed as

$$\mathbf{y} = \mathbf{W}^T \cdot \mathbf{x}, \tag{5}$$

where the column vector $\mathbf{y}$ is the feature vector in the projected space corresponding to pattern $\mathbf{x}$. The optimum matrix $\mathbf{W}$ is obtained by maximizing the criterion function

$$J(\mathbf{W}) = \mathbf{S}_{BF} / \mathbf{S}_{WF}, \tag{6}$$

where $\mathbf{S}_{BF}$ and $\mathbf{S}_{WF}$ are the corresponding scatter matrices in the (feature) projection space. It can be shown that $\mathbf{S}_{BF}$ and $\mathbf{S}_{WF}$ can be written as

$$\mathbf{S}_{BF} = \mathbf{W}^T \mathbf{S}_B \mathbf{W}$$
$$\mathbf{S}_{WF} = \mathbf{W}^T \mathbf{S}_W \mathbf{W}. \tag{7}$$

Therefore, $J(\mathbf{W})$ can be represented in terms of the scatter matrices of the original patterns

$$J(\mathbf{W}) = \frac{\mathbf{W}^T \mathbf{S}_B \mathbf{W}}{\mathbf{W}^T \mathbf{S}_W \mathbf{W}}. \tag{8}$$

$J(\mathbf{W})$ is a vector valued function, and the determinant of this function can be used as a scalar measure of $J(\mathbf{W})$. The columns of $\mathbf{W}$ that maximize the determinant of $J(\mathbf{W})$ are then the eigenvectors that correspond to the largest eigenvalues in the generalized eigenvalue equation:

$$\mathbf{S}_B \mathbf{w}_i = \lambda_i \mathbf{S}_W \mathbf{w}_i. \tag{9}$$

For nonsingular $\mathbf{S}_W$, Equation 9 can be written as

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w}_i = \lambda_i \mathbf{w}_i. \tag{10}$$

From Equation 10, we can directly compute the eigenvalues λ_i and the eigenvectors $\mathbf{w_i}$, which then constitute the columns of the $\mathbf{W}$ matrix.

Figure 6 illustrates the transformation of axes obtained by LDA, which takes the class information into consideration. LDA is not without its own limitations, however. A close look at the rank properties of the scatter matrices show that regardless of the dimensionality of the original pattern, LDA transforms a pattern vector onto a feature vector, whose dimension can be at most C-1, where C is the number of classes. This restriction is serious because, for problems with high dimensionality where most of the features do in fact carry meaningful information, a reduction to C-1 dimensions may cause loss of important information. Furthermore, the approach implicitly assumes that the class distributions are unimodal (such as Gaussian). If the original distributions are multimodal or highly overlapping, LDA becomes ineffective. In general, if the discriminatory information lies within different means (that is, centers of classes are sufficiently far away from each other), the LDA technique works well. If the discriminatory information lies within the variances, than PCA works better then the LDA.

A variation of LDA, referred to as nonparametric discriminant analysis (3,4) removes the unimodal assumption as well as the restriction of projection to a C-1 dimensional space, which is achieved by redefining the between-class matrix, making it a full rank matrix.

$$\mathbf{S}_B = \frac{1}{N} \sum_{i=1}^{C} \sum_{j=1}^{C} \sum_{\mathbf{x} \in X_i} w_{ijx} (\mathbf{x} - \mathbf{m}_{ijx}) \cdot (\mathbf{x} - \mathbf{m}_{ijx})^T, \tag{11}$$

where N is the total number of instances, $\mathbf{m}_{ijx}$ represents the mean of $\mathbf{x_i}$'s k-nearest neighbors from class ω_j and w_{ijx} represents the weight of the feature vector $\mathbf{x}$ from class ω_i to class ω_j

$$w_{ijx} = \frac{\min(\text{dist}(\mathbf{x}^i_{KNN}), \text{dist}(\mathbf{x}^j_{KNN}))}{\text{dist}(\mathbf{x}^i_{KNN}) + \text{dist}(\mathbf{x}^j_{KNN})}, \qquad (12)$$

with $\text{dist}(\mathbf{x}^i_{KNN})$ being the Euclidean distance from $\mathbf{x}$ to its k-nearest neighbors in class ω_i. In general, if a class ω_i instance is far away in the feature space from the cluster of class ω_j instances, w_{ijx} is a small quantity. If, however, an instance of class ω_i is close to the boundary of class ω_j instances, then w_{ijx} is a large quantity. The rest of the analysis is identical to that of regular LDA, where the generalized eigenvalue equation is solved to obtain the transformation matrix **W**. Those columns of **W** representing the eigenvectors of the largest eigenvalues are then retained and the remaining are discarded to achieve the desired dimensionality reduction.

3.3.3. Other Feature Extraction Techniques. Although PCA and LDA are very commonly used, they are not necessarily the best ones for all applications. In fact, depending on the application, the better discriminating information may reside in the spectral domain for which a Fourier-based transformation DISCRETE/FAST FOURIER TRANSFORM may be more appropriate. For NONSTATIONARY SIGNAL (such as ECG, EEG) a WAVELET-based time-frequency representation may be the better feature extraction technique (5). In such cases, the dimensionality reduction is obtained by discarding the transformation coefficients that are smaller than a threshold, assuming that those coefficients do not carry much information.

We should add that some sources treat the transformation-based dimensionality reduction techniques as a special case of feature selection, as the end result is the selection of a fewer number of features. These techniques, when considered as feature selection techniques, are referred to as *filtering*-based feature selection (as opposed to *wrapper*-based feature selection as described in the next section), as they *filter out* irrelevant features.

3.4. Feature Selection

In feature selection, this author specifically means selection of m features that provide the most discriminatory information, out of a possible d features, where $m < d$. In other words, by feature selection, this author refer, to selecting a subset of features from a set of features that have already been identified by a preceding feature extraction algorithm. The main question to answer under this setting is then "which subset of features provide the most discriminatory information?"

A criterion function is used to assess the discriminatory performance of the features, and a common choice for this function is the performance of a subsequent classifier trained on the given set of features. In essence, we are looking for a subset of features that leads to the best generalization performance of the classifier when trained on this subset. It should be noted, of course, the best subset then inevitably becomes a function of the classifier chosen. For example, the best subset of features for a neural network may be different than the one for a decision tree type classifier. The feature selection is therefore said to be *wrapped around* the classifier chosen, and, hence, such feature selection approaches are referred to as wrapper approaches (6).

There is, of course, a conceptually trivial solution to this problem: evaluate every subset of features by training a classifier with each such subset, observing its generalization performance, and then selecting the subset that provides the best performance. Such an *exhaustive search*, as conceptually simple as it may be, is prohibitively expensive (computation wise) even for a relatively small number of features. This is because, the number of subsets of features to be evaluated grows combinatorially as the number of features increase. For a fixed size of d and m, the number of subsets of features is $C(d,m) = d!/m!(d-m)!$. If, on the other hand, we are not just interested in a subset of features with cardinality m, but rather a subset whose cardinality is no larger than m, then the total number of subsets to be evaluated becomes $\sum_{i=1}^{m} C(d,i)$. Just for illustrative purposes, if we are interested to find the best set of features (of any cardinality) out of 12 features, 4095 subsets of features need to be evaluated. Or, if we are interested in finding the best subset with 10 features or less, out of 20 features, we would have to evaluate 184,756 subsets of features. It is not unusual for practical problems to have hundreds, if not thousands, of features.

Of course, more efficient search algorithms exists that avoid the full exhaustive search, such as the well-established depth-first search, breath-first search, branch and bound search, as well as hill climb search; however, each has it own limitations. For example, the branch and bound algorithm avoids the exhaustive search by searching subspaces and computing upper and lower bounds on the solutions obtained in these subspaces. The algorithm keeps track of the performance of the best solution found thus far, and if the performance of another solution is worse than the current best, the subspaces of this solution are discarded from future search. This of course, makes sense if and only if the criterion function used to evaluate the performance is monotonic on the feature subsets, that is, the performance of any feature subset must improve with the addition of features. As discussed earlier, this is clearly not the case in classification problems, as addition of irrelevant features are guaranteed to cause performance degradation.

Other search algorithms include sequential forward and backward search (also referred to as hill-climbing) (7). Forward search starts with no features and evaluates every single feature, one at a time, and finds the best single feature. This feature is then included as part of the optimal feature subset. Then, keeping this one feature, all other features are added, again one at a time, to form a two-feature subset. The best two-feature subset is retained and the search continues by adding one feature at a time to the best subset found thus far. The search terminates when addition of a new feature no longer improves performance. The backward search works much

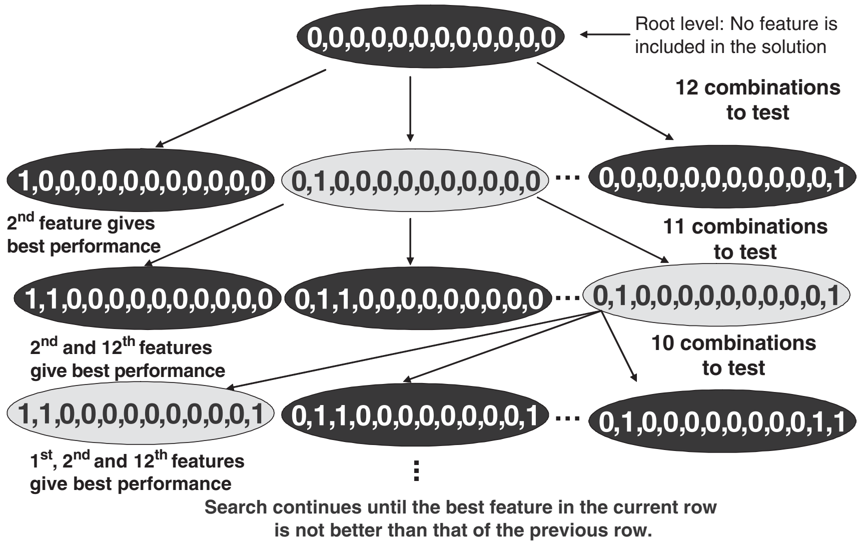


Figure 7. Forward search hill climb.

the same, but in reverse order: The search starts with all features and then one feature is removed at a time until removing features no longer improves performance.

Figure 7 illustrates the forward-search-based hill climb on a sample feature set of cardinality 12. Each ordered 12-tuple indicates which features are included or not (indicated by "1" or "0," respectively) in that feature subset. First row shows the root level where no features are yet included in the search. Each feature is evaluated one at a time (twelfth total evaluations). Assume that the second feature provides the best performance. Then, all two-feature combinations including the second feature are evaluated (11 such evaluations). Assume that second and 12th features together provide the best performance, which is better than the performance of the second feature alone. Then, keeping these two features, one more feature is added and the procedure continues until adding a feature no longer improves the performance from the previous iteration.

Forward and backward searches are significantly more cost efficient than the exhaustive search, as at each stage the number of possible combinations that need to be evaluated decreases by one. Therefore, the total number of feature subsets that must be evaluated is

$$N + (N+1) + (N+2) + \cdots + (N-(N-1)) \\ = \frac{N(N-1))}{2}, \tag{13}$$

where N is the total number of features. In comparison to 4095 exhaustive search evaluations, hill climb requires only 66 for the 12-feature example mentioned above. However, computational efficiency comes at the cost of optimality: Hill climb is suboptimal, as the optimal feature subset that provides the best performance need not contain the single best feature obtained earlier.

Despite their suboptimality, these search algorithms are often employed if (1) the total number of features is significantly large, and (2) the features are uncorrelated. We would like to emphasize the second condition: Wrapper-based feature selection algorithms can only be used if the individual features are uncorrelated and, better yet, if they are independent of each other. Such approaches cannot be used for time-series-based features, such as ECG signals, where one feature is clearly and strongly correlated with the features that come before and after itself.

4. MODEL SELECTION AND TRAINING

4.1. Setting the Problem as a Function Approximation

Only after acquiring and preprocessing adequate and representative data and extracting and selecting the most informative features is one finally ready to select a classifier and its corresponding training algorithm. As mentioned earlier, one can think of the classification as a function approximation problem: find a function that maps a d-dimensional input to appropriately encoded class information (both inputs and outputs must be encoded, unless they are already of numerical nature). Once the classification is cast as a function approximation problem, a variety of mathematical tools, such as optimization algorithms, can be used. Some of the more common ones are described below. Although most common pattern recognition algorithms are categorized as statistical approaches vs. neural network type approaches, it is possible to show that they are infact closely related and even a one-to-one match between certain statistical approaches and their corresponding neural network equivalents can be established.

4.2. Statistical Pattern Recognition

4.2.1. Bayes Classifier. In statistical approaches, data points are assumed to be drawn from a probability distribution, where each pattern has a certain probability of belonging to a class, determined by its class conditioned probability distribution. In order to build a classifier, these distributions must either be known ahead of time or must be learned from the data. The problem is cast as follows: A given d-dimensional$\mathbf{x} = (x_1, \ldots, x_d)$ needs to be assigned to one of c classes $\omega_1, \ldots, \omega_c$. The feature vector $\mathbf{x}$ that belongs to class ω_j is considered as an observation drawn randomly from a probability distribution conditioned on class ω_j, $P(\mathbf{x}|\omega_j)$. This distribution is called the *likelihood*, the (conditional) probability of observing a feature value of $\mathbf{x}$, given that the correct class is ω_j. All things being equal, the category with higher class conditional probability is more "likely" to be the correct class. All things may not always be equal, however, as certain classes may be inherently more likely. The likelihood is therefore converted into a *posterior probability* $P(\omega_j|\mathbf{x})$, the (conditional) probability of correct class being ω_j, given that feature value $\mathbf{x}$ has been observed. The conversion is done using the well-known Bayes theorem that takes the prior likelihood of each class into consideration:

$$P(\omega_j|\mathbf{x}) = \frac{P(\mathbf{x} \cap \omega_j)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\omega_j) \cdot P(\omega_j)}{\sum_{k=1}^{C} P(\mathbf{x}|\omega_k) \cdot P(\omega_k)}, \tag{14}$$

where $P(\omega_j)$ is the *prior probability*, the previously known probability of correct class being class ω_j (without regarding the observed feature vector), and $P(\mathbf{x})$ is the *evidence*, the total probability of observing the feature vector $\mathbf{x}$. The prior probability is usually known from prior experience. For example, referring to our original example, if historical data indicates that 20% of all people over the age of 60 have had two or more heart attacks, regardless of weight and blood pressure, the prior probability for this class is 0.2. If such prior experience is not available, it can either be estimated from the ratio of training data patients that fall into this category, or if that is not reliable (because of small training data), it can be taken as equal for all classes.

The posterior probability is calculated for each class, given $\mathbf{x}$, and the final classification then assigns $\mathbf{x}$ to the class for which the posterior probability is largest, that is, ω_i is chosen if $P(\omega_i|\mathbf{x}) > P(\omega_j|\mathbf{x})$, $\forall i, j = 1, \ldots, c$. It should be noted that the evidence is the same for all classes, and hence its value is inconsequential for the final classification. A classifier constructed in this manner is usually referred to as a *Bayes classifier* or Bayes decision rule, and can be shown to be the optimal classifier, with smallest achievable error, in the statistical sense.

A more general form of the Bayes classifier considers the fact that not all errors are equally costly, and hence tries to minimize the expected risk $R(\alpha_i|\mathbf{x})$, the expected loss for taking action α_i. Whereas taking action α_i is usually considered as choosing class ω_i, *refusing to make a decision* can also be an action, allowing the classifier not to make a decision if the expected risk of doing so is smaller than that of choosing any of the classes. The expected risk can be calculated as

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j) \cdot P(\omega_j|\mathbf{x}), \tag{15}$$

where $\lambda(\alpha_i|\omega_j)$ is the loss incurred for taking action α_i when the correct class is ω_j. If one associates action α_i as selecting ω_i, and if all errors are equally costly the *zero-one loss* is obtained

$$\lambda(\alpha_i|\omega_j) = \begin{cases} 0, & \text{if } i = j \\ 1, & \text{if } i \neq j. \end{cases} \tag{16}$$

This loss function assigns no loss to correct classification and assigns a loss of 1 to misclassification. The risk corresponding to this loss function is then

$$R(\alpha_i|\mathbf{x}) = \sum_{\substack{j \neq i \\ j=1,\ldots,c}} P(\omega_j|\mathbf{x}) = 1 - P(\omega_i|\mathbf{x}), \tag{17}$$

proving that the class that maximizes the posterior probability minimizes the expected risk.

Out of the three terms in the optimal Bayes decision rule, the evidence is unnecessary, the prior probability can be easily estimated, but we have not mentioned how to obtain the key third term, the likelihood. Yet, it is this critical likelihood term whose estimation is usually very difficult, particularly for high dimensional data, rendering Bayes classifier impractical for most applications of practical interest. One cannot discard the Bayes classifier outright, however, as several ways exist in which it can still be used: (1) If the likelihood is known, it is the optimal classifier; (2) if the form of the likelihood function is known (e.g., Gaussian), but its parameters are unknown, they can be estimated using MAXIMUM LIKELIHOOD ESTIMATION (MLE); (3) even the form of the likelihood function can be estimated from the training data, for example, by using k-nearest neighbor approach (discussed below) or by using Parzen windows (1), which computes the superposition of (usually Gaussian) kernels, each of which are centered on available training data points—however, this approach becomes computationally expensive as dimensionality increases; and (4) the Bayes classifier can be used as a benchmark against the performance of new classifiers by using artificially generated data whose distributions are known.

4.2.2. Naïve Bayes Classifiers. As mentioned above, the main disadvantage of the Bayes classifier is the difficulty in estimating the likelihood (class-conditional) probabilities, particularly for high dimensional data because of the curse of dimensionality. There is highly practical solution to this problem, however, and that is to assume class-conditional independence of the features:

$$P(\mathbf{x}|\omega_i) = \prod_{j=1}^{d} P(x^{(j)}|\omega_i), \tag{18}$$

which yields the so-called Naïve Bayes classifier. Equation 18 basically requires that the jth feature of instance $\mathbf{x}$, denoted as $x^{(j)}$, is independent of all other features, given the class information. It should be noted that this is not as nearly restrictive as assuming full independence, that is,

$$P(\mathbf{x}) = \prod_{j=1}^{d} P(x^{(j)}). \tag{19}$$

The classification rule corresponding to the Naïve Bayes classifier is then to compute the *discriminant function* representing posterior probabilities as

$$g_i(\mathbf{x}) = P(\omega_i)\prod_{j=1}^{d} P(x^{(j)}|\omega_i) \tag{20}$$

for each class i, and then choosing the class for which the discriminant function $g_i(\mathbf{x})$ is largest. The main advantage of this approach is that it only requires univariate densities $p(x^{(j)}|\omega_i)$ to be computed, which are much easier to compute than the multivariate densities $p(\mathbf{x}|\omega_i)$. In practice, Naïve Bayes has been shown to provide respectable performance, comparable with that of neural networks, even under mild violations of the independence assumptions.

4.2.3. k-Nearest Neighbor (kNN) Classifier. Although the kNN can be used as a nonparametric density estimation algorithm (see NEAREST NEIGHBOR RULES), it is more commonly used as a classification tool. As such, it is one of the most commonly used algorithms, because it is very intuitive, simple to implement, and yet provides remarkably good performance even for demanding applications. Simply put, the kNN takes a test instance $\mathbf{x}$, finds its k-nearest neighbors in the training data, and assigns $\mathbf{x}$ to the class occurring most often among those k neighbors. The entire algorithm can be summarized as follows:

- Out of N training vectors, identify the k-nearest neighbors (based on some distance metric, such as Euclidean) of $\mathbf{x}$ irrespective of the class label. Choose an odd (or prime) k.
- Identify the number of samples k_j that belongs to class ω_j such that $\sum_j k_j = k$.
- Assign $\mathbf{x}$ to the class with the maximum number of k_j samples.

Figure 8 illustrates this procedure for $k = 11$. According to the kNN rule, the test instance indicated by the plus sign ✚ would be labeled as class-3 (represented by circles ●), as out of its 11 nearest neighbors, class-3 instances occur most often in the training data [six instances, as opposed to two from class-2 (▲), three from class-4 (★), and none from class-1 (■)]. Similarly, the test instance indicated by the diamond shape ◆ would be labeled as class-1, because its 11 nearest neighbors have seven instances from class-1 and only four instances from class-4 (none from other classes) in the training data. It should be noted that the choice of k as an odd number is not by accident. For two-class problems, it is often chosen as an odd number, and for c-class problems, it is usually chosen as a number that is not divisible by c to prevent potential ties.

The limiting case of kNN is to chose $k = 1$, essentially turning the classification algorithm to one that assigns the test instance to the class of its NEAREST NEIGHBOR in the training data. Choosing a large k has the advantage of creating smooth decision boundaries. However, it also has higher computational complexity, but more importantly,

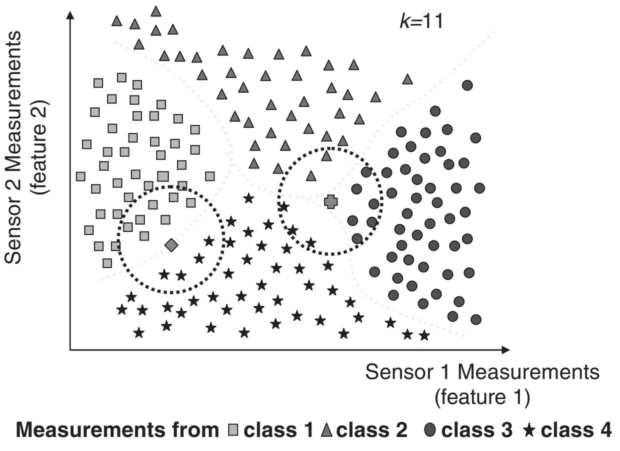


Figure 8. kNN classification.

loses local information because of averaging caused by further away instances being taken into account in the classification.

Despite its simple structure, the kNN is a formidable competitor to other classification algorithms. In fact, it can be easily shown that when sufficiently dense data exists, its performance approaches to that of the optimal Bayes classifier. Specifically, in the large sample limit, the error of the 1-NN classifier is at worst twice the error of the optimal Bayes classifier.

The kNN does not really do much of any learning. In fact, it does no processing until a request to classify an unknown instance is made. It simply compares the unknown data instance with those that are in the training data, for which it must have access to the entire database. Therefore, this approach is also called lazy learning or memory-based learning, which is in contrast to that of “eager” learning algorithms, such as neural networks, which do in fact learn the decision boundary before it is asked to classify an unknown instance. In eager learning algorithms, the training data can be discarded after the training: Once the classifier model has been generated, all information contained in the training data is then condensed and stored as model parameters, such as the neural network weights. Lazy algorithms have little or no computational cost of training, but more computational cost during the testing compared with eager learners.

4.3. Neural Networks

Among countless number of neural network structures, There are two that are used more often than all others: the multilayer perceptron (MLP) and the radial basis function (RBF). These networks have been extensively studied, empirically tested on a broad spectrum of applications, and hence their properties are now well known. Furthermore, these two types of networks have proven to be universal approximators (8–11), a term referring to the ability of these networks to approximate any decision boundary of arbitrary dimensionality and arbitrary complexity with arbitrary precision, given adequate amount of data and proper selection of their architectural and training pa-

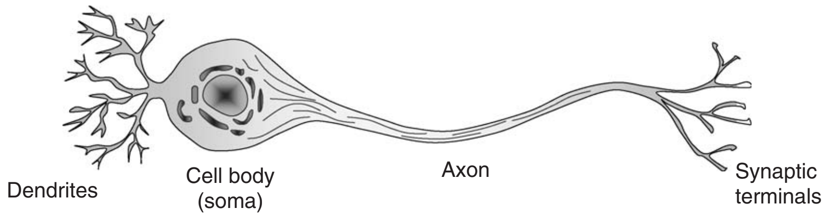


Figure 9. Schematic illustration of the neuron.

rameters. These two types of networks are explained in detail in this article.

4.3.1. Neuronal Model and the Multilayer Percept-rorn. The artificial neural networks (ANNs), or simply neural networks, are designed to mimic the decision-making ability of the brain, and hence their structure resembles that of the nervous system, albeit very crudely, featuring a massively interconnected set of elements. In its most basic form, a single element of the nervous system, a neuron, consists of four basic parts, as schematically illustrated in Fig. 9: the dendrites that receive information from other neurons, the cell body (the soma) that contains the nucleus and processes the information received by the neuron, the axon that is used to transmit the processed information away from the soma and toward its intended destination, and the synaptic terminals that relay this information to/from the dendrites of consecutive neurons, the brain, or the intended sensory/motor organ.

The neuron, modeled as the unit shown in Fig. 10, is typically called a node. It features a set of inputs and a weight for each input representing the information arriving at the dendrites, a processing unit representing the cell body and an output that connects to other nodes (or provides the output) representing the synaptic terminals. The weights, positive or negative, represent the excitatory or inhibitory nature of neuronal activation, respectively. The axon is implicitly represented by the physical layout of the neural network.

It is noted that for a d-dimensional feature vector, the node usually has $d+1$ inputs to the node often exist, where the 0th input is indicated as x_0 with the constant value of 1, and its associated weight w_0. This input serves as the bias of the node. The output of the node is calculated as the weighted sum of its inputs, called the *net sum*, or simply *net*, passed through the activation function *f*:

$$net = \sum_{i=0}^{d} w_{ji} x_i = \mathbf{x} \cdot \mathbf{w}^T + w_0 \qquad y = f(net). \tag{21}$$

Note that the *net sum* creates a linear decision boundary, $\mathbf{x} \cdot \mathbf{w}^T + w_0$, which is then modified by the activation function. Popular choices for the activation function *f* include

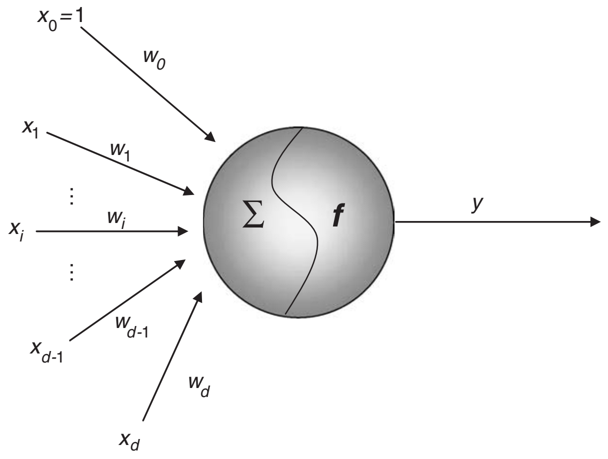


Figure 10. The node model of a neuron.

(1) The thresholding function

$$f(net) = \begin{cases} 1, & \textit{if net} \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad \text{or} \quad f(net) = \begin{cases} 1, & \textit{if net} \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{22}$$

When used with this activation function, the node is also known as the *perceptron*.

(2) The identity (linear) function

$$f(net) = net. \tag{23}$$

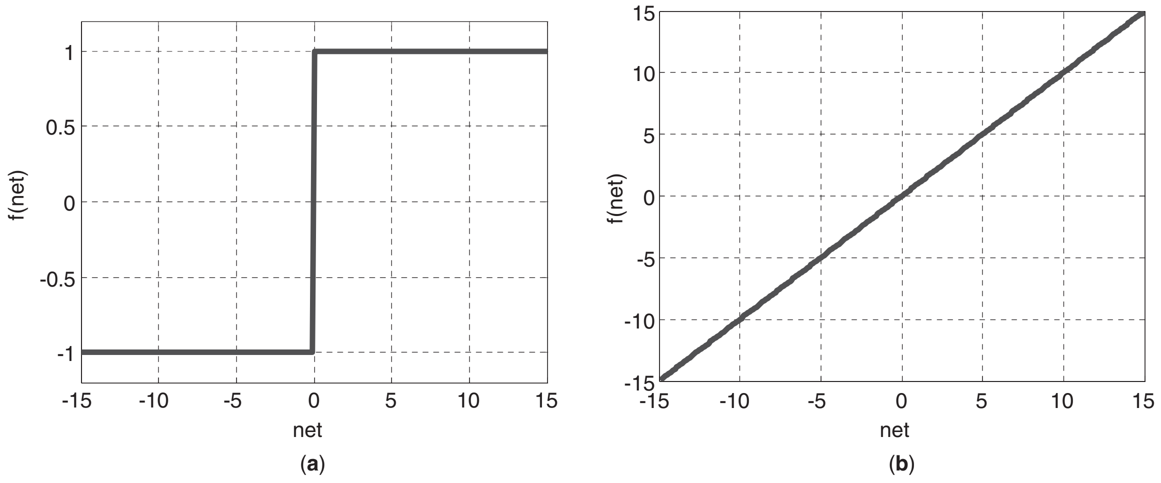
(3) The logistic (sigmoid) function

$$f(net) = \frac{1}{1+e^{-\beta \cdot net}}. \tag{24}$$

(4) The hyperbolic tangent (sigmoid) function

$$f(net) = \tanh(\beta net) = \frac{2}{1+e^{-\beta \cdot net}} - 1. \tag{25}$$

The thresholding function, the identity function, and the logistic sigmoid function are depicted in Figs. 11a 11b,



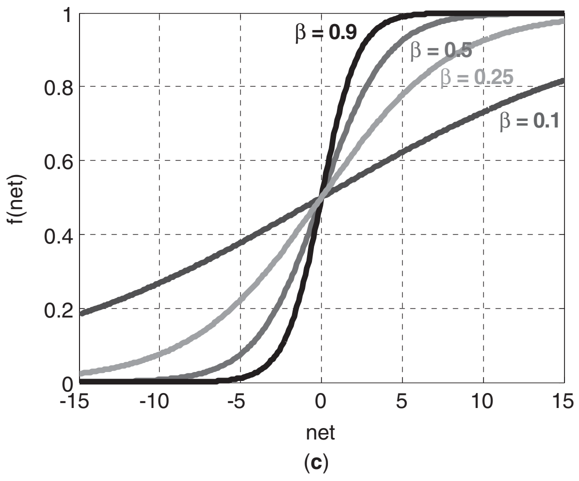


Figure 11. (a) The thresholding, (b) identity, and (c) logistic sigmoid activation functions.

and 11c, respectively (the shape of the hyperbolic tangent function looks similar to that of the logistic sigmoid, except the logistic function takes values between 0 and 1, whereas the range of the hyperbolic tangent is between -1 and 1). The β parameter in the sigmoidal functions controls the sharpness of the function transition at zero, and both functions approach to thresholding function as β approaches 1, and to a linear function as β approaches 0.

In the case of the thresholding function, the node simply makes a hard decision to fire (an action potential) or not, depending on whether it has a net excitatory or inhibitory weighted input. In the case of the identity function, the node does no processing and relays its input to its output, and in the case of the sigmoid function, it provides a continuous output in the range of [0 1] as a soft decision on whether to fire or not. Rosenblatt, who has coined the term perceptron, has shown that perceptron can learn any linear decision boundary through a simple learning al gorithm, where the weights are randomly initialized and then iteratively updated as

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t) \Rightarrow \Delta\mathbf{w} = \eta\varepsilon_i\mathbf{x}_i \tag{26}$$

for each $\mathbf{x_i}$ that is misclassified by the algorithm in the previous iteration, where η is the so-called *learning rate* and ε_i is the error of the perceptron for input $\mathbf{x_i}$. However, if the classes are not linearly separable, than the algorithm loops infinitely without convergence on a solution. As most problems have nonlinear decision boundaries, the single perceptron is of limited use.

Although a single perceptron may not be of much practical use, appropriate combinations of them can be quite powerful and approximate an arbitrarily complex nonlinear decision boundary. Arguably the most popular of classifiers constructed in such a fashion is the ubiquitous multilayer perceptron (MLP). The structure of the MLP is designed to mimic that of the nervous system: a large number of neurons connected together, and the information flows from one neuron to others in a cascade-like structure until it reaches its intended destination (Fig. 12).

Figure 13 provides a more detailed representation of the MLP structure on which we identify many architectural properties of this network. An MLP is a *feed-forward* type neural network, indicating that the information flows

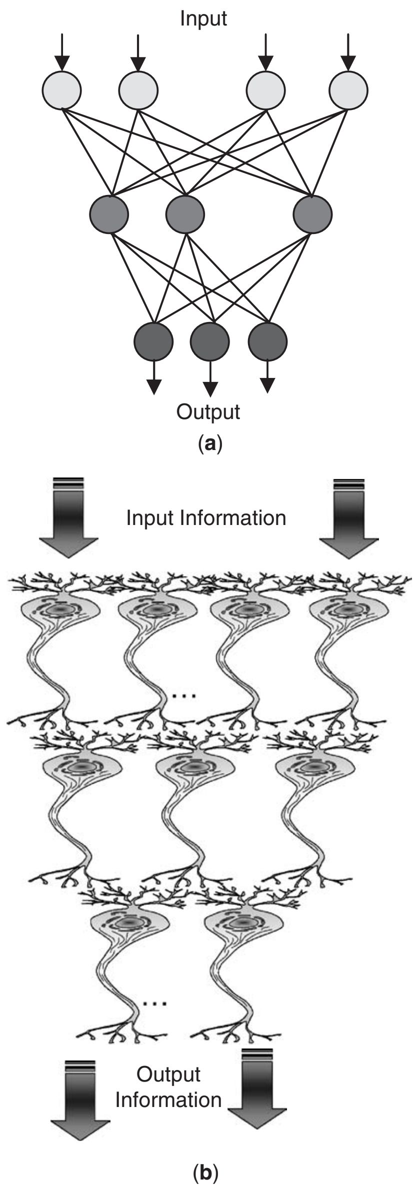


Figure 12. (a) The MLP structure mimicking (b) the simplified structure of the nervous system.

in one direction from the input toward the output (as opposed to *recurrent networks* that have feedback loops). The first layer, also called the *input layer*, uses nodes with linear activation functions, where each node has a single input corresponding to one of the features of the input vector. Therefore, the input layer consists of d nodes, for a d-dimensional feature vector (an additional input node with a constant value of 1 is also routinely added to serve as the bias term), As a result of the linear activation functions, the input layer does no processing apart from presenting the data to the next layer, called the *hidden layer*. An MLP may have one or more hidden layers; however, it can be shown that any decision boundary function of arbitrary dimensionality and arbitrary complexity can be realized with a single hidden layer MLP. Each input node is fully connected to every other node in the following layer through the set of weights $\mathbf{W_{ji}}$. The hidden layer nodes use a nonlinear activation function, typically a sigmoid function. The outputs of the hidden layer nodes are then fully connected either to the next hidden layer's nodes, or more commonly to the *output layer nodes* through another set of weights, $\mathbf{W_{kj}}$. The output layer nodes, of which one exists for each class, also use a sigmoidal activation function. The class labels of the training data are encoded using c binary digits (e.g., class-3 in a 5-class problem is represented with [0 0 1 0 0], whereas class-5 is represented as [0 0 0 0 1]). The logistic function forces each output to be between 0 and 1. The value for each output is then interpreted as the *support* given by the MLP to the corresponding class. In fact, if a sufficiently dense training dataset is available, if the MLP architecture is chosen appropriately to learn the underlying data distribution, and if the outputs are normalized to add up to 1, then individual outputs approximate the posterior probability of the corresponding class. The *softmax* rule is typically used for normalizations (1,9): representing the actual classifier output corresponding to the kth class as $z_k(\mathbf{x})$, and the normalized values as $z'_k(\mathbf{x})$, approximated posterior probabilities $P(\omega_k|\mathbf{x})$ can be obtained as

$$P(\omega_k|\mathbf{x}) \approx z'_k(\mathbf{x}) = \frac{e^{z_k(\mathbf{x})}}{\sum_{c=1}^{C} e^{z_c(\mathbf{x})}} \Rightarrow \sum_{k=1}^{C} z'_k(\mathbf{x}) = 1. \qquad (27)$$

The decision is therefore chosen as the class indicated by the output node that yields the largest value for the given input (that is, choosing the class with largest posterior probability).

Although the number of nodes for the input and output layers are fixed (number of features and classes, respectively), the number of nodes for the hidden layer is a free parameter of the algorithm. In general, the predictive power of the MLP—its ability to approximate complex decision boundaries—increases with the number of hidden layer nodes, however, so does its chance of overfitting the data, not to mention its memory and computational burden. Therefore, the least number of nodes that provide a desirable performance should be used (per Ockham's razor).

The goal of the MLP is to adjust its weights so that a cost function calculated as the squared error on labeled training data is minimized. This is achieved by using an optimization algorithm, typically one of several gradient-descent-based approaches, where the algorithm tries to find the global minimum of the so-called quadratic error surface defined by the cost function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(t_k - z_k)^2, \qquad (28)$$

where $\mathbf{w}$ represents the entire set of weights of the MLP, N is the number of training data instances, c is the number of classes, t_k is the target (correct) output for the kth output node, and z_k is the actual output of the kth output node. The algorithm by which the weights are iteratively

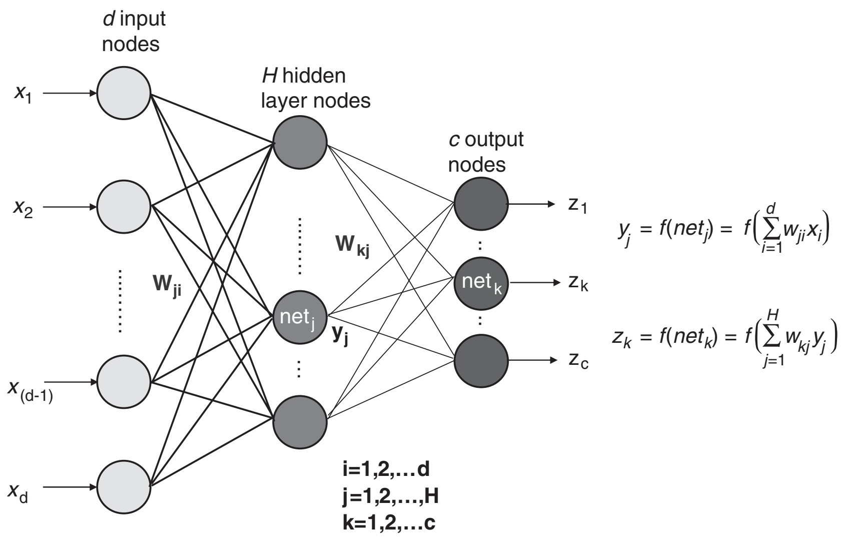


Figure 13. The MLP architecture.

learned is called the BACKPROPAGATION LEARNING RULE where the weight update searches for the minimum of the error surface along the direction of the negative gradient of the error:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t) \Rightarrow \Delta\mathbf{w} = -\eta\frac{\partial J(\mathbf{w})}{\partial\mathbf{w}}. \tag{29}$$

The careful reader will immediately notice that the cost function in Equation 28 does not appear to be a function of the weights **w**. However, the cost function is in fact an implicit function of the weights, because the values of the output nodes depend on all network weights. The algorithm begins with random initialization of all weights, and then calculates the weight update rule for the output weights first. As the actual outputs as well as the desired target outputs are available for the output layer, it can be easily shown to yield

$$\Delta w_{kj} = \eta\cdot\delta_k\cdot y_j = \eta(t_k - z_k)f'(net_k)y_j, \tag{30}$$

where the selection of logistic sigmoid as the activation function yields the convenient

$$f'(net_k) = f(net_k)[1 - f(net_k)] \tag{31}$$

as the derivative of the activation function, and where

$$\delta_k(t_k - z_k)f'(net_k) \tag{32}$$

is called the *sensitivity* of the *k*th output node.

Representing J as a function of input layer weights is a little trickier, because the desired outputs of the hidden layer nodes are not known. The trick itself is not a difficult one, however, and involves *backpropagating the error*, that is, representing the error at the output as a function of the input to hidden layer weights through a series of chain rules, yielding

$$\begin{aligned}\Delta w_{ji} &= -\eta\frac{\partial J}{\partial w_{ji}} = \eta\left[\sum_{k=1}^{c}\delta_k w_{kj}\right]f'(net_j)x_i\\ &= \eta\cdot\delta_j\cdot x_i,\end{aligned} \tag{33}$$

where

$$\delta_j = \left[\sum_{k=1}^{c}\delta_k w_{kj}\right]f'(net_j) \tag{34}$$

is called the sensitivity of the *j*th hidden layer node.

Successful implementation of the MLP requires suitable and appropriate choices of its many parameters. These parameters include the learning rate η, the number of hidden layer nodes H, the error goal E (the value of the cost function below which we consider the network as trained), the activation function, the way in which the weights are initialized, and so on. Although no hard rules exist for choosing any of these parameters, the follows guidelines are often recommended.

4.3.1.1. Learning Rate. The learning rate represents the size of each iterative step taken in search of the minimum error. The learning rate, in theory, only affects the convergence time. However, in practice, too large a value of learning rate can also result in system divergence, as the globally optimum solution is seldom found. A too small choice of η results in long training time, whereas a too large choice can cause the algorithm to take too big steps and miss the minimum of the error surface. Typical values of η lie in the [0 1] range.

4.3.1.2. Number of Hidden Layer Nodes H. As mentioned earlier, H defines the expressive power of the network, and larger H results in a network that can solve more complicated problems. However, excessive number of hidden nodes causes overfitting, the phenomenon where the training error can be made arbitrarily small, but the

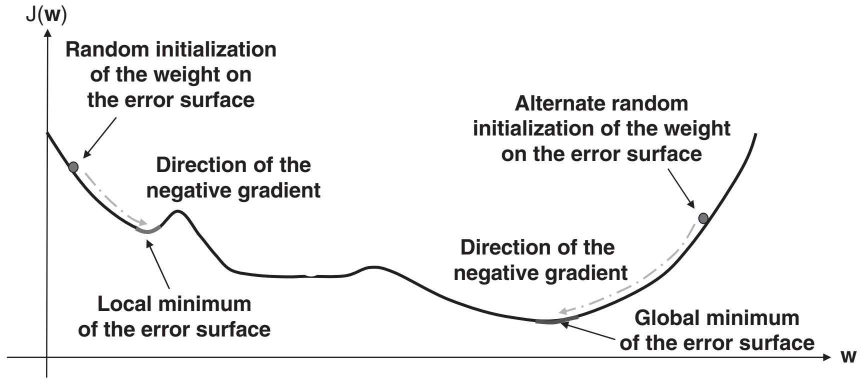

J(**w**)
Random initialization of the weight on the error surface
Direction of the negative gradient
Local minimum of the error surface
Alternate random initialization of the weight on the error surface
Direction of the negative gradient
Global minimum of the error surface
w

Figure 14. Schematic illustration of the local minima problem in gradient-descent-type optimization.

network performance on the test data remains poor. Too few hidden nodes, however, may not be enough to solve the more complicated problems. Typically, H is determined by a combination of previous expertise, amount of data available, dimensionality, complexity of the problem, trial and error, or validation on an additional training dataset. A common rule of thumb that is often used is to choose H such that the total number of weights remains less then $N/10$, where N is the total number of training data available.

4.3.1.3. Error Goal E. If the training data is noisy or overlapping, reaching zero error is usually not possible. Therefore, an error goal is set as a threshold, below which training is terminated. Selecting E too large causes premature termination of the algorithm, whereas choosing it too small forces the algorithm to learn the noise in the training data. E is also selected based on prior experience, trial and error, or validation on an additional training dataset, if one is available.

4.3.1.4. Choice of Activation Function and Output Encoding. If the problem is one of classification, then the output node activation function is typically chosen as a sigmoid. If the desired output classes are binary encoded (e.g., [0 0 1 0 0] indicating class-3 in a 5-class problem), then logistic sigmoid is used. Plus or minus 1 can also be used to encode the outputs (e.g., $[-1\ -1\ +1\ -1\ -1]$), in which case the hyperbolic tangent is employed. It has been empirically shown, however, that better performance may be obtained if softer values are used instead of the asymptotic ones, such as 0.05 instead of 0 and $\pm$ 0.95 instead of $\pm$ 1 during training. If the MLP is being used strictly for function approximation, linear activation function is used at the output layer, although the radial basis function (discussed below) is preferred for such function approximation problems.

4.3.1.5. Momentum. Small plateaus or local minima in the error surface may cause backpropagation to take a long time, or even get stuck at a local minimum, as schematically illustrated in Fig. 14, for the leftmost selection of initial weights. In order to prevent this problem, a momentum term is added, which incorporates the speed in which the weights are learned. This term is loosely related to the momentum in physics—a moving object keeps moving unless prevented by outside forces. The modified weight update rule is then obtained as follows

$$\boldsymbol{w}(t+1)=\boldsymbol{w}(t)+(1-\alpha)\Delta\boldsymbol{w}(t)+\alpha\Delta\boldsymbol{w}(t-1), \qquad (35)$$

where α is the momentum term. For $\alpha=0$, the original backpropagation is obtained: The weight update is determined purely by the gradient descent. For $\alpha=1$, the gradient is ignored, and the update is based on the momentum: The weight update continues along the direction in which it was moving previously. Although α is typically taken to be in [0.9 0.95] range, an inappropriate choice may lead to oscillations in training. The optimal value is often problem dependent and can be determined by trying several values and choosing the one that leads to smallest overall error.

Figure 14 shows that the minimum error reached also depends on the weight initialization. If, for example, the rightmost location had been randomly selected, the gradient descent would have easily located the global minimum of the error surface. Although an effective use of the momentum term reduces the dependence of the final solution to the initialization of the weights, it is common practice to train the MLP several times, with different random initializations of the weights, and choosing the one that leads to the smallest error.

4.3.2. Radial Basis Function (RBF) Networks. As pointed out earlier, the classification problem can also be cast as a function approximation problem, where the goal of the classifier is to approximate the function that provides an accurate mapping of the input patterns to their correct classes. There are many applications, however, where the problem strictly calls for a function approximation (also called system identification), or regression, particularly when the output to be determined is not one of several categorical entities, but rather a number on a discrete or continuous scale. For example, one may be interested in estimating the severity of dementia, on a scale of 0 to 10, from a series of electroencephalogram (EEG) measurements, or estimating the cerebral blood oxygenation levels obtained through near-infrared spectroscopy measurements. Clearly, because the output is not categorical, it is not a classification problem, but rather a function approximation: we assume that there is an unknown function that maps the features obtained from the signal to a number that represents the sought after value, and it is

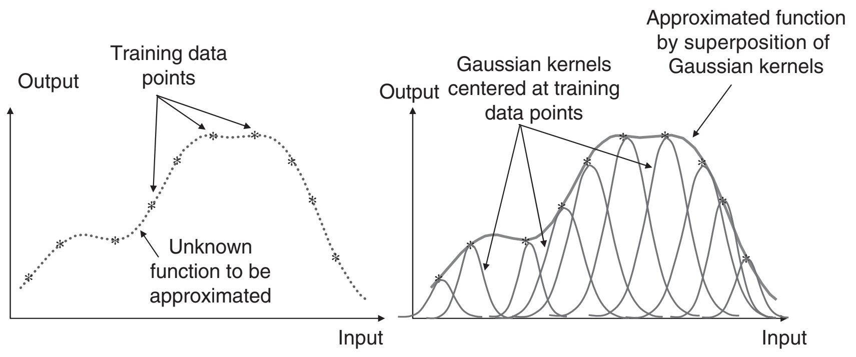


Figure 15. Function approximation using radial basis functions.

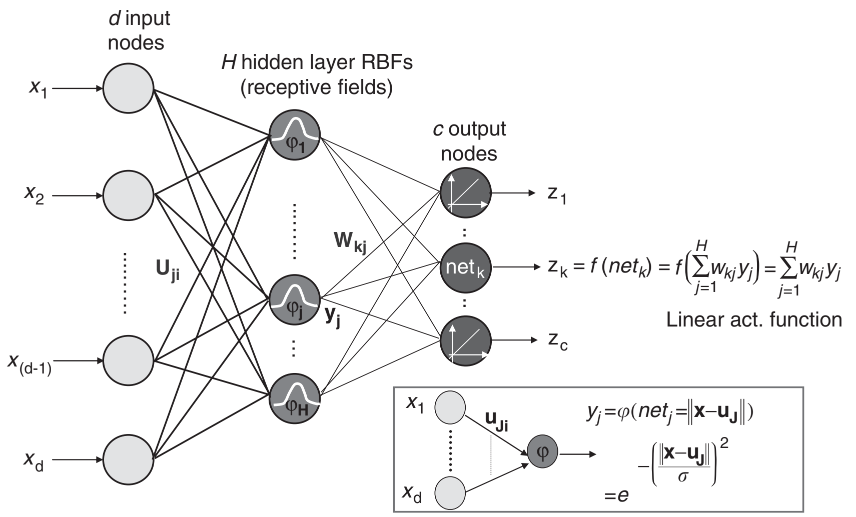

d input nodes
H hidden layer RBFs (receptive fields)
c output nodes
x_1
x_2
$x_{(d-1)}$
x_d
U_{ji}
W_{kj}
φ_1
φ_j
φ_H
y_j
net_k
z_1
$z_k = f(net_k) = f\left(\sum_{j=1}^{H} w_{kj} y_j\right) = \sum_{j=1}^{H} w_{kj} y_j$
Linear act. function
z_c
x_1
x_d
u_{Ji}
φ
$y_j=\varphi(net_j=\|x-u_J\|)$
$=e^{-\left(\frac{\|x-u_J\|}{\sigma}\right)^2}$

Figure 16. The RBF network architecture.

this function we wish to approximate from the given training data. It is for such function approximation applications that the RBF networks are most commonly used.

We should note that both the MLP and the RBF can be used for either classification or function approximation applications, as they are both universal approximators. However, many empirical studies have shown that the RBF networks usually perform well on function approximation problems, and MLP networks usually perform well on classification problems. As discussed below, the structure of the RBF network makes it a natural fit for function approximation.

Given a set of input/output pairs as training data drawn from an unknown (to be approximated) function, the main idea in RBF network is to place *radial basis functions*, also called *kernel functions*, on top of each training data point, and then compute the superposition of these functions to obtain an estimate of the unknown function. Figure 15 illustrates the approach, where the dotted line on the left indicates a 1D unknown function and the stars indicate the training data obtained from this function. A Gaussian-type radial basis function is then placed on top of each training data point (right). The inputs of the training data determine the centers, whereas their desired values determine the amplitudes of the kernel functions. Superposition of these kernel functions then provides an estimate of the unknown function.

The term *radial basis functions* is an appropriately chosen one: *radial* refers to the symmetry of the kernel, whereas *basis functions* are those functions using a linear combination of which any other function (in the same function space) can be represented. The RBF network automates the above described procedure using the architecture shown in Fig. 16. RBF is also a feed-forward network, and its architecture shows great resemblance to that of the MLP.

The main difference between the RBF and the MLP networks is the activation function used in the hidden and output layers. At the hidden layer, the RBF uses the radial basis functions for activation, typically the Gaussian. The hidden layer of the RBF is also referred to as the receptive field layer. The output of the jth receptive field is calculated as

$$\begin{aligned} y_j &= \varphi(net_j = \| \mathbf{x} - \mathbf{u_j} \|) \\ &= e^{-\left(\frac{\|\mathbf{x}-\mathbf{u_j}\|}{\sigma}\right)^2}, \end{aligned} \tag{36}$$

where φ is the radial basis function; $\mathbf{u_j}$ are the input to receptive field weights, which also constitute the centers of the Gaussian kernels, and are usually obtained as prototypes of the training data; and σ, the main free parameters of the algorithm, is the spread (standard deviation) of the Gaussian kernel. The spread should be chosen judiciously: Large σ values result in wider Gaussians, which can be helpful in averaging out the noise in the data. However, local information can also be lost because of this averaging. Small σ values, on the other hand, result in narrow Gaussians, which help retain the local information, however, may cause spurious peaks where training data is not very dense, particularly if those areas have noisy data instances.

The hidden layer units essentially compute the distance between the given input and preset centers, called *prototypes* (determined by the **u** weights), and calculates a support from each kernel for the given input: The support is large from those kernels whose centers are close to the given input, and small for others. The kth output node is the linear combination of these supports:

$$z_k = f(net_k) = f\left(\sum_{j=1}^{H} w_{kj} y_j\right) = \sum_{j=1}^{H} w_{kj} y_j = \mathbf{y} \cdot \mathbf{w}^T, \qquad (37)$$

where the activation function is usually the linear function, H is the number of receptive field units, and **w** is the set of weights connecting the receptive layer to the output layer.

An RBF network can be trained in one of four ways:

4.3.2.1. Approach 1. Approach 1 is known as exact RBF, as it guarantees correct classification of all training data instances. It requires N hidden layer nodes, one for each training instance. No iterative training is involved. RBF centers (**u**) are fixed as training data points, spread as variance of the data, and **w** are obtained by solving a set of linear equations. This approach may lead to overfitting.

4.3.2.2. Approach 2. In order to reduce the overfitting, fewer fixed centers are selected at random from the training data, where $H<N$. Otherwise, same as Approach 1, with no iterative training.

4.3.2.3. Approach 3. In this approach, centers are obtained from unsupervised learning (clustering). Spreads are obtained as variances of clusters, and **w** are obtained through LEAST MEAN SQUARES (LMS) algorithm. Both clustering and LMS algorithm are iterative, and therefore the approach is more computationally expensive than the previous ones. However, this approach is quite robust to overfitting, typically providing good results. It is the most commonly used procedure.

4.3.2.4. Approach 4. All unknowns are obtained from gradient-based supervised learning. The most computationally expensive approach, and with the exception of certain specialized applications, the additional computational burden do not necessarily translate into better performance.

In comparing MLP and RBF, their main common denominator is that they are both universal approximators. They have several differences, however: (1) MLP generates a global approximation to the decision boundaries, as opposed to the RBF, which generates more local approximations; (2) MLP is more likely to battle with local minima and flat valleys than RBF, and hence, in general, has longer training times; (3) as MLPs generate global approximations, they excel in extrapolating, that is, classifying instances that are outside of the feature space represented by the training data. Extrapolating, however, may mean dealing with outliers; (4) MLPs typically require fewer parameters than RBFs to approximate a given function with the same accuracy; (5) all parameters of an MLP are trained simultaneously, whereas RBF parameters can be trained separately in an efficient hybrid manner; (6) RBFs have a single hidden layer, whereas MLPs can have multiple hidden layers; (7) the hidden neurons of an MLP compute the inner product between an input vector and the weight vector, whereas RBFs compute the distance between the input and the radial basis function centers; (8) the hidden layer of an RBF is nonlinear and its output layer is usually linear, whereas all layers of an MLP are nonlinear. This really is more of a historic preference based on empirical success that MLPs typically do better in classification type problems and RBFs do better in function approximation type problems.

5. PERFORMANCE EVALUATION

Once the model selection and training is completed, its generalization performance needs to be evaluated on previously unseen data to estimate its true performance on field data.

One of the simplest, and hence, the most popular methods for evaluating the generalization performance is to split the entire training data into two partitions, where the first partition is used for actual training and the second partition is used for testing the performance of the algorithm. The performance on this latter dataset is then used as an estimate of the algorithm's true (and unknown) field performance. However, estimating the algorithm's future performance in this manner still presents some legitimate concerns:

1. As a portion of the dataset is set aside for testing, not all of the original data instances are available for training. Since an adequate and representative training data set is of paramount importance for successful training, this approach can result in suboptimal performance.
2. What percent of the original data should be set aside for testing? If a small portion is used for testing, then the estimate of the generalization performance may be unreliable. If a large portion of the dataset is used for testing, less data can be used for training leading to poor training.
3. We can also never know whether the test data that is set aside, regardless of its size, is a representative of the data the network will see in the future. After all, the instances in the test dataset may have a large distribution nearby or far away from the decision boundary, making the test dataset too difficult or too simple to classify, respectively.

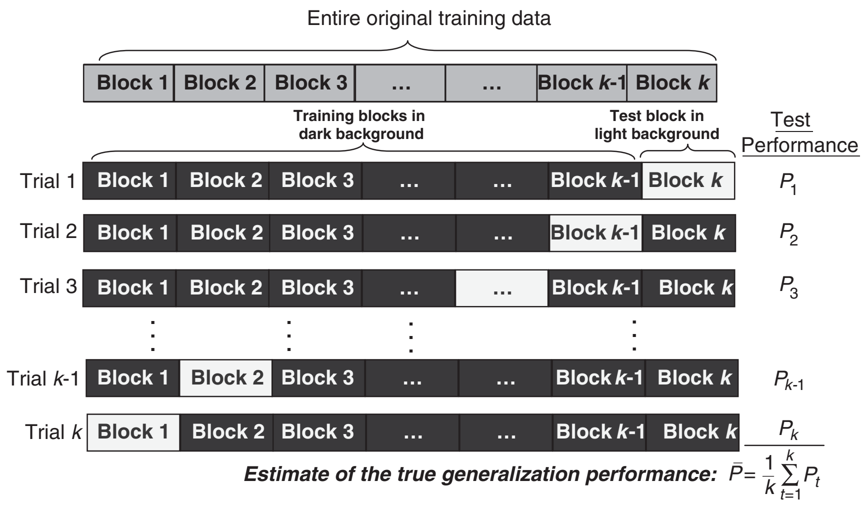

Entire original training data
Block 1
Block 2
Block 3
…
Block k-1
Block k
Training blocks in dark background
Test block in light background
Test Performance
Trial 1
Trial 2
Trial 3
Trial k-1
Trial k
P_1
P_2
P_3
P_{k-1}
P_k
Estimate of the true generalization performance: $\bar{P}=\frac{1}{k}\sum_{t=1}^{k}P_t$

Figure 17. k-fold cross validation.

One approach that has been accepted to provide a reasonably good estimate of the true generalization performance uses cross validation to address the above mentioned issues. In *k-fold cross validation*, the entire available training dataset is split into $k > 2$ partitions, creating k blocks of data. Of these k blocks, $k - 1$ are used for training and the remaining kth block is used for testing. This procedure is then repeated k times, using a different block for testing in each case. The k test performances obtained are averaged, and the average test performance is declared as the estimate of the true generalization performance of the algorithm. The procedure is illustrated in Fig. 17.

The algorithm has several important advantages. First, all data instances are used for training as well as for testing, but never at the same time, allowing full utilization of the data. Second, as the procedure is repeated k times, the probability of an unusually lucky or unlucky partitioning is reduced through averaging. Third, a confidence interval for the true generalization performance can be obtained using a two-tailed *z-test*, if $k > 30$, or a two-tailed *t-test*, otherwise.

For example, the $100^*(1-\alpha)\%$ two-sided confidence interval *t-test* would provide

$$P = \overline{P} \pm t_{\alpha/2,k-1} \cdot \frac{s}{\sqrt{k}}, \tag{38}$$

where $\overline{P}$ is the mean of k individual test performances $P_1 \sim P_k$, s is the standard error of k individual performance values (can be estimated by calculating the standard deviation), $t_{\alpha/2,k-1}$ is the critical value of t-distribution for $k-1$ degrees of freedom for the desired confidence level α, and P is the range of values in which the true generalization performance is considered to lie.

The choice of k is of course data dependent: selecting k too large divides the data into too many partitions, allowing a larger amount of data for training; however, it also increases the confidence interval because of large variations on the test performances, as a result of testing on very few test instances. On the other hand, choosing k too small causes not enough partitions to be made and reduces the amount of data available for training. For sufficiently large datasets, k is typically taken as 5 or 10; however, for smaller datasets, k may be chosen larger to allow a larger training data. The extreme case, where $k = N$, is also known as leave-one-out validation. In this case, the entire data but one instance is used for each training session, and testing solely on the remaining instance, repeating the process N times. Although this method provides the best estimate of the generalization performance, it usually yields a wides worst confidence interval, not to mention additional computational burden.

6. OTHER TOPICS & FUTURE DIRECTIONS

6.1. Other Common Pattern Recognition Approaches

As this chapter is meant to provide an introductory background to pattern recognition, only those topics that are most fundamental, and only those algorithms that are most commonly used have been described in the previous paragraphs. The field of pattern recognition, however, comprises of a vast number of algorithms, many of which are specifically geared toward certain applications falling outside the scope of this article. Nevertheless, a few additional well-established algorithms should be mentioned to provide breadth and topical name recognition.

As a preprocessing step, a technique that has some similarities to principal component analysis, but intended for solving a different problem is the INDEPENDENT COMPONENT ANALYSIS (ICA). Unlike PCA, which seeks to find directions in feature space that best represent the data in mean square error sense, ICA seeks to find directions along which the data components are most independent from each other (12). Typically used for BLIND SOURCE SEPARATION applications (13), ICA attempts to decouple different components of a composite data, which are originally generated by different sources. Isolating electrooculogram (caused by eye-blink artifacts) from an EEG, or isolating fetus ECG from that of the mother's are traditional applications for ICA.

As mentioned in the introduction, this chapter focuses on supervised algorithms, for which the availability of a

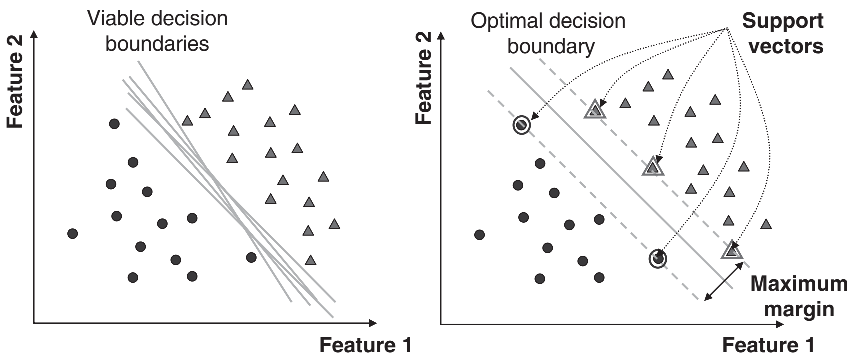


Figure 18. Support vectors as maximum margin classifiers.

training data with pre-assigned class labels is assumed. If such a training dataset is not available, unsupervised clustering algorithms can be used (1,2). Most commonly used examples of such algorithms include the k-means (1), several variations of fuzzy c-means (14), vector quantization (8,15) adaptive resonance theory (ART) networks (16), Kohonen networks (8,15) and variations of self-organizing maps (SOMs) (15).

In supervised learning, several prominent approaches are also commonly used, such as time-delay neural networks (17), recurrent networks (such as Hopfield network) (8), neuro-fuzzy algorithms (18), ARTMAP networks (19), learning vector quantization (8), genetic algorithms (20), swarm intelligence (21,22), and decision-tree-based approaches (1,23), among others. Also within the realm of supervised learning, multiple classifier (ensemble) systems and kernel-machine-based approaches (such as support vector machines) are rapidly gaining momentum, which are discussed in more detail below as current and developing research areas.

6.2. Current and Developing Research Areas

6.2.1. Kernel-Based Learning and Support Vector Machines. An area that has seen much recent interest is kernel-based learning with particular emphasis on the support vector machines (SVM). Introduced by Vapnik, SVMs (also called *maximum margin classifiers*) try to find the optimal decision boundary by maximizing the margin between the boundaries of different classes (24–26), as shown in Fig. 18. To do so, SVMs identify those instances of each class that define the boundary of that class in the feature space. These instances, considered to be the most informative ones, are called the *support vectors*, which are calculated through a quadratic programming-type constrained optimization algorithm.

Identifying margin defining instances may be easy on the simple 2D illustrative example of Fig. 18 where the classes are linearly separable, but SVMs really shine in their ability to find such instances in more complex, linearly nonseparable high dimensional spaces. In fact, one of the main novelties of SVMs is their ability to use a kernel function to map the feature space into a higher dimensional space, where the classes are linearly separable, and find the support vectors in that high dimensional space. Furthermore, what makes SVMs even more popular is the fact that they do so without actually requiring any calculations in the higher dimensional space, through the procedure affectionately known as the *kernel trick*. The trick is figuratively illustrated in Figs. 19a and 19b, and on the famous XOR problem in Figs. 19c and 19d.

Using SVMs with several different computationally less expensive optimization algorithms, implementing the algorithm for multiclass problems are ongoing research areas.

6.2.2. Mixture Models and Ensemble Learning. Another rapidly developing area within the pattern recognition community is the so-called *ensemble systems*, also known under various names such as multiple classifier systems (MCS), multiple expert models, or mixture models. The idea behind ensemble systems is to employ several classifiers, instead of a single classifier, to solve the particular pattern recognition problem at hand (27). Several interesting properties of ensemble systems have been established over the last several years that have made them quite popular. For example, Schapire has shown that using an algorithm called boosting (28), and later **AdaBoost** (29), *a weak learner*, an algorithm that generates classifiers that can merely do better than random guessing can be turned into a *strong learner* that generates a classifier that has an arbitrarily small error, with a guarantee that the combined system will perform as good or better than the best individual classifier that makes up the ensemble. Several variations of such ensemble systems can be seen in hierarchical mixture of experts (HME) (30) that use the **expectation maximization** (EM) (31) algorithm for training, or in stacked generalization (32) that uses a second-level classifier that is trained on the outputs of the first level classifiers. It has been well-established that much is to be gained from combining classifiers if the classifiers are as independent or as *diverse* as possible. That way, the individual weakness or instability of each classifier can be effectively averaged out by the combination process (such as voting-based approaches), which may significantly improve the generalization performance of the classification system. A specific area of interest within the ensemble systems researchers involves developing effective techniques for measuring diversity among classifiers, as well as investigating various combination schemes.

More recently, ensemble-systems-based approaches have also been proposed for such applications as incremental learning, data fusion, and confidence estimation using the **Learn**$^{++}$ algorithm (33). Incremental learning

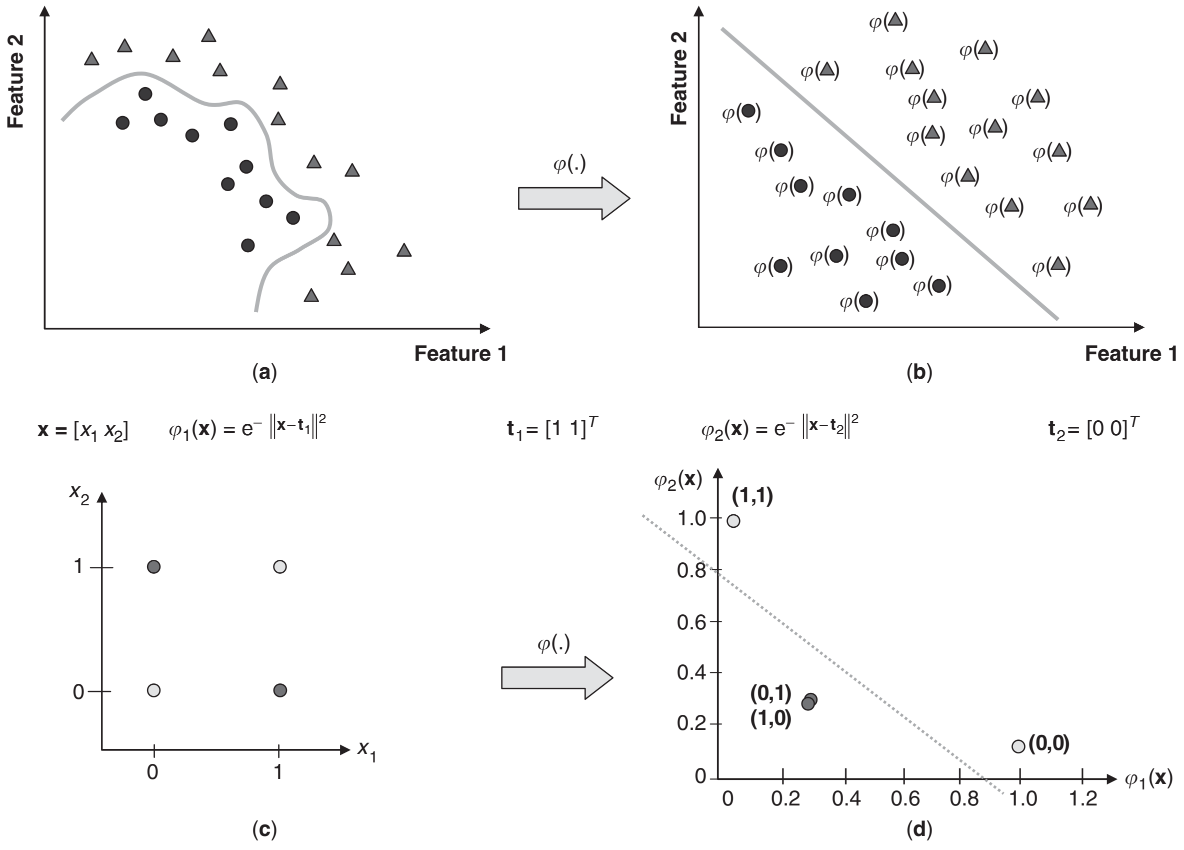


Figure 19. (a) Nonlinearly separable two-class problem (b) made linearly separable with the use of a kernel function. (c) Nonlinearly separable XOR problem (d) made linearly seaprable with the use of an exponential kernel.

is defined as the ability of a classifier to learn additional information from new data that may later become available, even if the new data introduce instances from previously unseen classes and even if the previous data are no longer available. **Learn**$^{++}$ generates an ensemble of classifiers for each dataset that becomes available, where the individual classifiers are specifically geared toward learning the novel information presented by the new data. The algorithm is also used for data fusion applications, where the additional data that become available may consist of a different set of features. Finally, as the ensemble systems employ several classifiers, estimating the confidence of classification system also becomes possible without necessarily resorting to cross-validation techniques.

6.3. Applications

A large number of applications benefit from current and near future research in pattern recognition. Some of these areas are in fact very closely related to active research areas in biomedical engineering. For example, computational neuroscience is one of the rapidly expanding areas, where researchers try to understand more intricate details of how the nervous system really works; propose new models of neurons and local neural circuits as well as new learning rules based on these discoveries. Spiking neural networks is one type of neural network that is actively being considered for this area.

Another area related to biomedical engineering is bioinformatics and genomics, where new and existing pattern recognition algorithms are evaluated in gene sequence analysis, development of artificial immune systems, and genomic data mining. Vision and image processing, biometric identification (face, fingerprint, iris scan, etc.), handwritten character recognition, auditory data and speech processing, speech and speaker identification, and diagnosis of various cardiovascular and neurological disorders based on features obtained from noninvasive tests or bioimaging modalities are all biomedical application-oriented areas of active research.

Among all biomedical engineering-related applications of pattern recognition, those that involve analysis of biological signals for detection or identification of certain pathological conditions have enjoyed the most attention. For example, analysis of various features extracted from the ECG to diagnose a broad spectrum of cardiovascular disorders, including arrhythmias and heart rate variability, have been popular areas of research. Using EEG signals to diagnose/detect various neurological conditions—from detection of characteristic EEG waves to advanced detection of epileptic conditions to automated early diagnosis of Alzheimer's diseases—have also enjoyed significant attention. Analysis of respiratory signals to detect pulmonary disorders such as apnea and hypopnea, analysis of Raman spectra for early detection of skin cancer, analysis of aging on gait patterns, analysis of various imaging modalities (such as MIR, PET scan, digital mammogram) for automated detection of tumors or other abnormalities are just a few of many applications of pattern recognition in biomedical engineering.

Of course, pattern recognition is also being actively used in nonbiomedical applications, such as general signal processing and telecommunications applications, robotics and dynamic control, financial or other time series data analysis, military and computer security applications, remote sensing and oceanographic applications, geographical information systems, nondestructive evaluation, hazardous compound identification based on chemical sensor data, among many others.

6.4. Some Final Thoughts

Literally hundreds of pattern recognition approaches and algorithms exist, and it is often asked whether any one of them is consistently better than the others. A cleverly named theorem, called the *no-free lunch theorem*, tells us that no algorithm is universally superior to all others in the absence of any additional information (34). In fact, it can be proven that problems exist for which random guessing will outperform any other algorithm. Although such problems may not be of much practical interest, there is still a lesson to be learned: The choice of the appropriate algorithm almost invariably depends on the nature of the problem, the distribution that provides the data for that problem, and the prior knowledge available to the designer.

6.5. For Further Reading

The nature of this chapter makes it impossible to provide a more detailed discussion on all topics, or to provide specific algorithms for all techniques. Therefore, the author's goal has been to provide a fundamental background, upon which interested readers can build. The references cited within the article will provide the additional depth and breadth; however, a topic-specific list is also provided below for textbook based references.

Pattern Classification (2nd edition), by Duda, Hart and Stork (Wiley, 2000); *Statistical Pattern Recognition* by Webb (Wiley, 2002), *Pattern Recognition* (2nd edition) by Theodoridis and Koutroumbas (Academic Press, 2003) and *The Elements of Statistical Learning* by Hastie, Tibshirani and Friedman (Springer, 2002) cover a broad spectrum of pattern recognition topics in considerable detail. For algorithms and techniques that are specifically related to neural networks, *Neural Networks, A Comprehensive Foundation* by Haykin (Prentice Hall, 1998), *Neural Networks for Pattern Recognition* by Bishop (Oxford University Press, 1995), and *Pattern Recognition and Neural Networks* by Ripley (Cambridge University Press, 1996) are classic texts that provide a very comprehensive theoretical coverage of neural network-based approaches. For the practitioners, *Netlab* by Nabney and *Computer Manual in Matlab to Accompany Pattern Classification* by Stork and Yom-Tov include Matlab functions for many of the popular pattern recognition algorithms, whereas *Neural and Adaptive Systems*: *Fundamentals through Simulations* by Principe, Euliano and Lefebvre (Wiley, 1999) and *Pattern Recognition*, *Concepts*, *Methods and Applications* by Marques de Sa (Springer, 2001) both include a simulation software on CD that allows the readers to simulate the algorithms discussed in their respective books. *Pattern Recognition* in *Medical Imaging* by Meyer-Baese (Academic Press, 2003) also covers most pattern recognition approaches along with feature extraction and signal/image processing techniques that are particularly applicable to large dimensional datasets, such as medical images. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* by Cristianini and Taylor (Cambridge University Press, 2000) provides an excellent introduction to kernel-based methods, whereas *Combining Pattern Classifiers*: *Methods and Algorithms* by Kuncheva (Wiley, 2004) is the only text of its kind covering ensemble-based systems. For a more light-hearted and recreational reading, *Intelligent Data Analysis*, *an Introduction* by Berthold and Hand (Springer, 2003) and *How to Solve It*: *Modern Heuristics*, by Michalewics and Fogel (Springer, 2000) can be recommended. Finally, also of interest is *PRTools* a set of pattern recognition functions organized as a Matlab toolbox, available from www.prtools.org.

A large number of journals, such as *IEEE Transactions in Pattern Analysis and Machine Learning, Neural Networks, Knowledge and Data Engineering, Fuzzy Systems, Systems Man and Cybernetics*, as well as *Transactions on Biomedical Engineering*, Elsevier's *Pattern Recognition, Neural Networks, Information Fusion*, *Neurocomputing* and many others publish papers on theory and applications of pattern recognition. For the most recent advances in pattern recognition, the readers may wish to attend some of the major conferences, such as the International Joint Conference on Neural Networks (IJCNN), International Conference on Artificial Neural Networks (ICANN), International Conference on Pattern Recognition (ICPR), Neural Information Processing Systems Conference (NISP), and International Workshop on Multiple Classifier Systems (MCS). All of these conferences feature sessions on biomedical applications, and all major biomedical engineering conferences, such as IEEE's Engineering in Medicine and Biology Conference, typically feature a healthy number of presentations on pattern recognition.

BIBLIOGRAPHY

1. R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. New York: Wiley, 2000.
2. A. K. Jain, R. P. W. Duin, and J. Mao, Statistical pattern recognition: a review. *IEEE Trans. Pattern Analysis Machine Intell*. 2000; **22**(1):4–37.
3. K. Fukunaga and J. Mantock, Nonparametric discriminant analysis. *IEEE Trans. Pattern Analysis Machine Intell*. 1983; **5**:671–678.
4. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. San Diego, CA: Academic Press, 1990.

5. M. Akay, ed. *Time Frequency and Wavelets in Biomedical Signal Processing*. Piscataway, NJ: IEEE Press, 1998.
6. R. Kohavi and G. H. John, Wrappers for feature subset selection. *Artific. Intell*. 1997; **97**(1–2):273–324.
7. A. Jain and D. Zongker, Feature selection: evaluation, application, and small performance. *IEEE Trans. Pattern Analysis Machine Intell*. 1997; **19**(2):153–158.
8. S. Haykin, *Neural Networks, A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1998.
9. C. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University, 1995.
10. D. Hush and B. Horne, Progress in supervised neural networks. *IEEE Signal Proc. Mag.* 1993; **10**(1):8–39.
11. A. Webb, *Statistical Pattern Recognition*, 2nd ed. New York: Wiley, 2002.
12. A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. New York: Wiley, 2001.
13. A. Hyvärinen and E. Oja, A fast fixed point algorithm for independent component analysis. *Neural Computation* 1997; **9**(7):1483–1492.
14. N. R. Pal, K. Pal, J. M. Keller, and J. Bezdek, A possibilistic fuzzy c-means clustering algorithm. *IEEE Trans. Fuzzy Syst*. 2005; **13**(4):517–530.
15. T. Kohonen, *Self Organizing Maps*. Berlin, Germany: Springer, 2000.
16. G. A. Carpenter and S. Grossberg, The ART of adaptive pattern recognition by a self-organizing neural network. *Computer* 1988; **21**(3):77–88.
17. A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, Phoneme recognition using time-delay neural networks. *IEEE Trans. Acoust. Speech Signal Proc*. 1989; **37**(3):328–339.
18. J. R. Jang, C. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*: *A Computational Approach to Learning and Machine Intelligence*. Englewood, NJ: Pearson, 1996.
19. G. A. Carpenter, S. Grossberg, S. Markuzon, J. H. Reynolds, and D. B. Rosen, Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Trans. Neural Networks* 1992; **3**(5):698–713.
20. M. Mitchell, *An Introduction to Genetic Algorithm* (*Complex Adaptive Systems*). Cambridge, MA: The MIT Press, 1998.
21. R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence*. Burlington, MA: Morgan Kauffman, 2001.
22. E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence*: *From Natural to Artificial Systems*. Cambridge, UK: Oxford University Press, 1999.
23. J. R. Quinlan, *C4.5*: *Programs for Machine Learning*. Burlington, MA: Morgan Kaufmann, 1992.
24. C. J. C. Burges, A tutorial on support vector machines for pattern recognition. *Data Mining Knowledge Discov.* 1998; **2**(2):121–167.
25. V. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. Berlin: Springer, 2000.
26. B. Scholkoph and A. Smola, *Learning with Kernels*: *Support Vector Machines*, *Optimization and Beyond*. Cambridge, MA: MIT Press, 2002.
27. L. I. Kuncheva, *Combining Pattern Classifiers*: *Methods and Algorithms*. New York: Wiley, 2004.
28. R. E. Schapire, The strength of weak learnability. *Machine Learn*. 1990; **5**:197–227.
29. Y. Freund and R. Schapire, A decision theoretic generalization of on-line learning and an application to boosting. *Comput. Syst. Sci*. 1997; **57**(1):119–139.
30. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, Adaptive mixtures of local experts. *Neural Computation* 1991; **3**:79–87.
31. M. I. Jordan and R. A. Jacobs, Hierarchical mixtures of experts and the EM algorithm. *Neural Computation* 1994; **6**(2):181–214.
32. D. H. Wolpert, Stacked generalization. *Neural Networks* 1992; **5**(2):241–259.
33. R. Polikar, L. Udpa, S. Udpa, and V. Honavar, Learn + +: an incremental learning algorithm for supervised neural networks. *IEEE Trans. Syst. Man Cybernet*. (*C*) 2001; **31**(4):497–508.
34. D. H. Wolpert and W. G. Macready, No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation* 1997; **1**(1):67–82.