Introduction to Pattern Recognition

Wei-Lun Chao
Graduate Institute of Communication Engineering
National Taiwan University, Taiwan

October, 2009

Abstract

Pattern recognition is not a new field of research, actually, theories and techniques about it has developed for a long time. While with the fast advancement of computer architecture, machine learning, and computer vision, computational complexity is possible to be dealt with and more and more new ways of thinking are brought into the research of pattern recognition. In this report, I'd like to introduce the basic concept, compact explanation, widely-used methods of pattern recognition, and some outstanding applications are included.

Content

1. Introduction

What's the meaning of pattern recognition? This is a question I asked myself after doing this research for five weeks. At first, I thought it is a kind of applied science and case-oriented, but the book chapters I read are mainly of probability, linear algebra, and lots of algorithms, which made me really confused about this topic. With continued studying, I gradually catch the basic ideas of it, and this report will be a summary of my studying.

Pattern recognition is a process that taking in raw data and making an action based on the category of the pattern. There are some interesting explanations about the term 'pattern'. Norbert Wiener [1] said that "A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made, rather than by the intrinsic nature of these elements", and Watanabe said that "A pattern is the opposite of a chaos; it is an entity vaguely defined, that could be given a name". After analyzing the raw data based on a certain purpose and method, we can do actions such as classification or clustering on those raw data. Classification is to classify a data into known categories, while clustering is to create new categories from a set of data. In the sense of training, classification is also called "supervised classification" and clustering as "unsupervised classification".

Human beings can easily recognize things or objects based on past learning experiences. For example, when we look at a picture with cars, roads, people, and building inside, we can tell which part is a person or a car, its shape, and even the brand of a car or the gender of a specific person. But what do computers do for recognition? Could they learn as what we did? Could they directly tell what inside a digital picture? If not, what could we do to give computers the ability as we have? This may be the motivation and purpose of the research of pattern recognition.

At early stages, researchers want to use math to analyze and model this process, and the Bayesian decision theory is the best classifier based on the view of statistics. But there're still some problems make Bayesian classifier difficult to implement or make its result imperfect, and other recognition techniques such as template matching and the syntactic method also face some limitations. Neural networks present another way of thinking. In contrast to define all the math terms by people, this technique only gives the classifier the basic structure, the modification methods, and the way to show the classification result, while remains the classifier parameters undefined. Recently, even many breakthroughs about pattern recognition are presented, there're still many problems unsolved. And that’s why this topic is still really hot now.

In following sections, I will explain the basic concepts and structure of pattern recognition, introduce each part, and present my understanding about it, finally with a conclusion. And in the rest part of this report, I’ll focus more pattern recognition cases on images, which is easier to understand.

2. Basic structure

Through several years of development, there has been a basic structure of pattern recognition, and methods of it can be classified into some categories. In this section I’ll introduce those basic concepts and hope to give readers a clear idea about how to build a recognition process.

2.1 Two factors of pattern recognition

When talking about pattern recognition, we may ask what to, and how to recognize, and these two things are two key factors in this field. For example, if I want the computer to recognize if there is a car in a picture, the thing to be recognized is a car. But how could a computer detect a car from a matrix with thousands of element and the value of each could have hundreds of possibilities (a digital picture is actually a 2-D array raw data).

Generally, we can separate the whole process as “Features” and “Classifiers”. A pattern is an arrangement of descriptors (features) [1]. Or we can say that from the example before, the car is the object (or pattern), and the features can be its shape (by boundary). Classifiers mean the mechanisms and methods to define what the pattern is after we get a set of data to describe it. The methods to describe a pattern depend on what we want to recognize and the applications, and there is a strong connection between the method of feature generation and the methods of classification. For example, if we describe a pattern as a structure, then we have to use a structure classification method to recognize it. The complexity of feature generation and classification is about complement, which means if the feature can faithful render the pattern, then the classifier could become easier, and vice versa. The simplest feature of image recognition is just the raw data.

2.2 basic structure of the system

The basic structure of a pattern recognition system is shown in fig 2.1. The classifier should contain the knowledge of each pattern category and also the criterion or metric to discriminate among patterns classes. The knowledge could be defined by people or trained from a set of training data, and the criterion is related to the description of pattern categories. Then when we want to assign an unknown-class pattern, we only have to extract its features and put the features inside the classifier and we can get the result.

2.3 Four basic recognition models

Processes of pattern recognition can be simply classified into four different models: Template matching, Syntactic, Statistical, and Neural Network approaches. The former two methods are introduced in section 4, while the latter two methods are in section 3. Table 2.1 shows some information about these four models [2]. From [1], there is another way to classify pattern recognition methods based on the description of patterns:

- **Quantitative description:** using length, measure of area, and texture, etc. to describe the pattern, and sometimes there is no relation between each component.
- **Structure descriptions:** using some qualitative factors to describe patterns. Strings and trees are two kinds of arrangements, and there exists order, permutation, or hierarchical relations between each component.

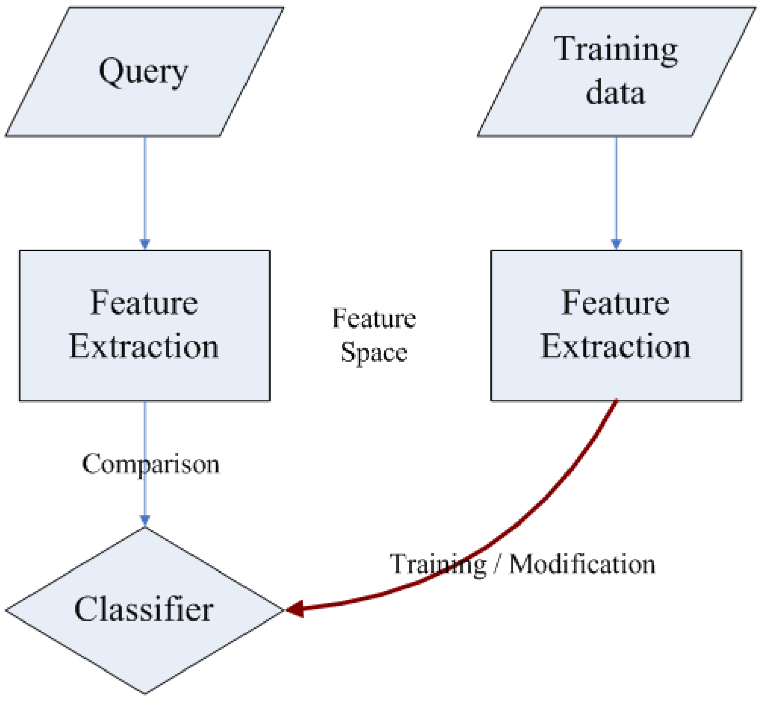
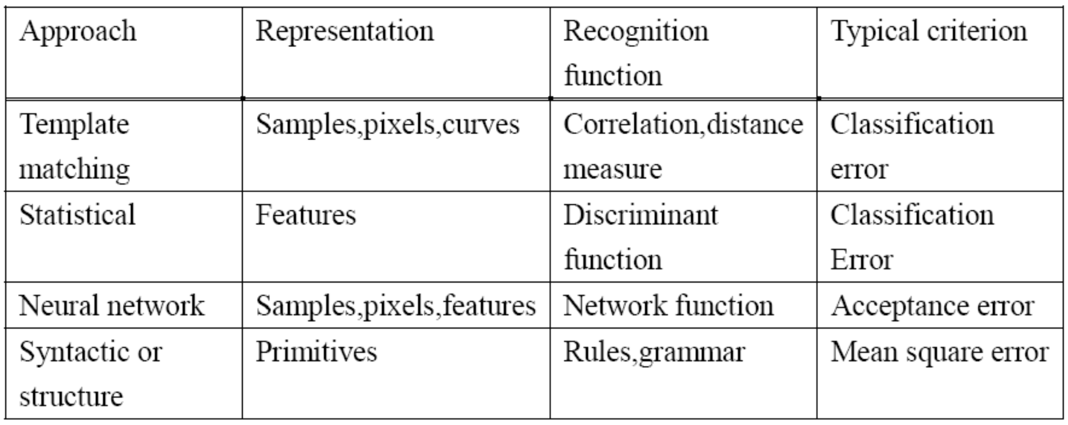


Figure 2.1: The basic structure of a pattern recognition system

Table 2.1: Pattern recognition models



Approach	Representation	Recognition function	Typical criterion
Template matching	Samples,pixels,curves	Correlation,distance measure	Classification error
Statistical	Features	Discriminant function	Classification Error
Neural network	Samples,pixels,features	Network function	Acceptance error
Syntactic or structure	Primitives	Rules,grammar	Mean square error

2.4 Applications of pattern recognition

Table 2.2: Some applications of pattern recognition (from the pattern recognition course website [3])

Problem to deal with	Input	Output
Identification and counting of cells	Slides of blood samples, micro-sections of tissues	Type of cells
Inspection (PC boards, IC masks, textiles)	Scanned image (visible, infrared)	Acceptable/unacceptable
Manufacturing	3-D images (structured light, laser, stereo)	Identify objects, pose, assembly
Web search	Key words specified by a user	Text relevant to the user
Fingerprint identification	Input image from fingerprint sensors	Owner of the fingerprint, fingerprint classes
Online handwriting retrieval	Query word written by a user	Occurrence of the word in the database
Speech recognition	Speech waveforms	Spoken words, speaker identity
Non-destructive testing	Ultrasound, eddy current, acoustic emission waveforms	Presence/absence of flaw, type of flaw
Detection and diagnosis of disease	EKG, EEG waveforms	Types of cardiac conditions, classes of brain conditions
Natural resource identification	Multispectral images	Terrain forms, vegetation cover
Aerial reconnaissance	Visual, infrared, radar images	Tanks, airfields
Character recognition (page readers, zip code, license plate)	Optical scanned image	Alphanumeric characters

3. Classification methods I

In this section, I will introduce some classification methods of quantitative descriptions. A pattern here is described as a vector, and there seems no necessary order between each element.

3.1 Description of a pattern category

At the beginning, I'd like to talk about the description of a pattern category. A pattern is described as a vector, then how about a category which may contain several patterns (ex. Training data) as a pattern class. Statistical moments may be a compact description, such as a mean vector, the covariance matrix, or even a distribution model. Other methods combine the description of pattern classes and the decision criterion together to build a set of decision functions or decision matrices.

An important view is about the relation between training data and class description (or even the decision criterion). A classifier learns from the training data and has the ability to recognize an unknown pattern, while a poor process of it could result in distortion. On the other hand, if the training data is not enough, even an excellent learning algorithm could produce a poor classifier. The goal of the combination of training data and learning algorithm is to build a classification mechanism as better as possible especially for a vector which is not one of the training data.

One of the most simplest classifiers is the look-up table method, while this method needs large storage space, and requires enough training data to cover all possible cases. This method doesn't have any ability to deal with a test pattern whose feature is not included in the classifier.

3.2 Decision-Theoretic Methods

Decision-theoretic approaches to recognition are based on the use of decision (or discriminant) functions. For an test vector $\boldsymbol{x}$ and $\boldsymbol{W}$ pattern classes $w_1, w_2, \ldots\ldots, w_W$, the basic problem in decision-theoretic pattern recognition is to find W decision functions $d_1(x), d_2(x), \ldots\ldots, d_W(x)$ with the property that, if pattern $\boldsymbol{x}$ belongs to class w_i, then:

$$d_i(\boldsymbol{x}) > d_j(\boldsymbol{x}) \quad j = 1,2, \ldots\ldots, W \quad j \neq i \tag{3.1}$$

or

$$d_{ij}(\boldsymbol{x}) = d_i(\boldsymbol{x}) - d_j(\boldsymbol{x}) > 0 \quad j = 1,2,\dots\dots,W \quad j \neq i \tag{3.2}$$

where $d_{ij}(x) = 0$ is the decision boundary separating class w_i from w_j. The goal of this subsection is to develop various approaches for finding decision functions that satisfy eq. (3.1).

3.2.1 Matching by minimum distance classification

Recognition techniques based on matching represent each class by a prototype pattern vector, and an unknown pattern is assigned to the class to which it is closet in terms of predefined metric. The minimum distance classifier is to measure the Euclidean distance between test vector and each of class vectors, and the smallest distance one is selected as the decision.

An easy way to define the prototype vector of each class is the mean vector of the patterns of each class. The smallest distance from the test vector to each mean vector implies the best match. The decision function to each class could be defined as:

$$d_j(\boldsymbol{x}) = \| \boldsymbol{x} - m_j \| = \boldsymbol{x}^T m_j - \frac{1}{2} m_j^T m_j \quad j = 1,2,\dots\dots,W \tag{3.3}$$

In practice, the minimum classifier works well when the distance between means is larger compared to the spread or randomness of each class with respect to its mean. This classifier yields optimum performance (in terms of minimizing the average loss of misclassification) when the distribution of each class about its mean is in the form of a spherical “hypercloud” in *n*-dimensional pattern space, while this kind of situation occurs seldom in practice unless the system designer controls the nature if input.

3.2.2 Matching by correlation

The spatial correlation between a mask $w(x,y)$ of size $m \times n$, with an image $f(x,y)$ may be expressed in the form:

$$c(x,y) = \sum_s \sum_t w(s,t) f(x+s, y+t) \tag{3.4}$$

where the limits of summation are taken over the region shared by *w* and *f*. Just as spatial convolution is related to the Fourier transform of the function via the convolution theorem, spatial correlation is related to the transforms of the functions via the correlation theorem:

$$f(x,y) \star w(x,y) \Leftrightarrow F^*(u,v) W(u,v) \tag{3.5}$$

Where ‘⋆’ indicates spatial convolution here. The c(x, y) is sensitive to scale changes in *f* and *w*, so the normalized correlation is used:

$$\gamma(x,y) = \frac{\sum_s \sum_t [w(s,t) - \bar{w}][f(x+s,y+t) - \bar{f}(x+s,y+t)]}{\{\sum_s \sum_t [w(s,t) - \bar{w}]^2 \sum_s \sum_t [f(x+s,y+t) - \bar{f}(x+s,y+t)]^2\}^{1/2}} \quad \gamma(x,y) \in [-1, 1] \tag{3.6}$$

where the limit of summation are taken over the region shared by *w* and *f*, $\bar{w}$ is the average value of the mask, and $\bar{f}(x+s, y+t)$ is the average value of *f* in the region coincident with *w*. Often, *w* is referred to as a template and correlation is referred to as template matching. The maximum value of $\gamma(x, y)$ occurs when the normalized *w* and the corresponding normalized region in *f* are identical.

From eq. (3.6), correlation for changes in intensity values of the functions being processed could be normalized. Normalizing for size and rotation is a more complicated problem, and additional information is needed to execute them.

3.2.3 Optimum Statistical Classifiers

This is a probabilistic approach to recognition. Probability considerations become important in pattern recognition because of the randomness under which pattern classes normally are generated. The classification is optimum in the sense that, on average, its use yields the lowest probability of committing classification errors.

The probability that a particular $\boldsymbol{x}$ comes from class w_i is denoted as $p(w_k/\boldsymbol{x})$. If the pattern classifier decides that $\boldsymbol{x}$ came from w_j when it actually came from w_i, it incurs a loss L_{ij}, and here the way for deciding L_{ij} is beyond consideration. The average loss incurred in assigning $\boldsymbol{x}$ to class w_j is:

$$\gamma_{\mathrm{j}}(\boldsymbol{x}) = \sum_{\mathrm{k}=1}^{\mathrm{W}} L_{kj}\, p(w_k/\boldsymbol{x}) \tag{3.7}$$

This equation often is called the conditional average risk or loss in decision-theory terminology. Eq. (3.10) can be further written in the form:

$$\gamma_j(\boldsymbol{x}) = \frac{1}{p(x)} \sum_{k=1}^{W} L_{kj}\, p(\boldsymbol{x}/w_k)\, p(w_k) \tag{3.8}$$

and some common terms can be dropped without affecting the relative order:

$$\gamma_j(\boldsymbol{x}) = \sum_{k=1}^{W} L_{kj}\, p(\boldsymbol{x}/w_k)\, p(w_k) \tag{3.9}$$

After all the $\gamma_j(x)$s for each class has been calculated, the smallest one will be the best decision, and the classifier that minimizes the total average loss is called the Bayes Classifier. A test $\boldsymbol{x}$ is assigned to class w_i if $\gamma_i(x) < \gamma_j(x)$, for all $j \neq i$.

If we further assume the loss for a correct decision is 0 and a failed decision is 1, then Eq. (3.7) can be further written as the form of decision function:

$$d_j(\boldsymbol{x}) = p\big(\boldsymbol{x}/w_j\big)p(w_j) \tag{3.10}$$

where a pattern vector *x* is assigned to the class whose decision function yields the largest numerical value.

However, the optimum case is built on the assumption that all the probability terms are known. There are two unknown terms in eq. (3.10), $p(\boldsymbol{x}/w_j)$ and $p(w_j)$, and usually we assume $p(w_j) = 1/W$ and Gaussian probability density function for $p(\boldsymbol{x}/w_j)$. In the n-dimensional case, the Gaussian density of the vector in the j-th pattern class has the form:

$$p(\boldsymbol{x}/w_j) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{C}_j|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_j)^T \boldsymbol{C}_j{}^{-1}(\boldsymbol{x}-\boldsymbol{m}_j)} \tag{3.11}$$

where each density is specified completely by its mean vector $\boldsymbol{m}_j$ and covariance matrix $\boldsymbol{C}_j = E_j\{(\boldsymbol{x}-\boldsymbol{m}_j)(\boldsymbol{x}-\boldsymbol{m}_j)^T\}$. For an optimum statistical classifier using Gaussian PDF model, mean m_j, covariance matrix C_j, and $p(w_j)$ are needed to know. Here we use a procedure, "Training" or "Learning", to estimate these quantities. First, we need some vectors which have been determined for specific classes, then use them to get the $\boldsymbol{m_j}$, $\boldsymbol{C_j}$, and even $p(w_j)$. More data for training, the result will be better but meanwhile taking more time. Keep in mind that optimality requires that the PDF and a priori probability of each class be known.

From the statement above, how to build the PDF is the key point of Bayes classifier. The process to get the PDF is composed of model selection first and training later, and sometimes the model may not fit the information from training data. The closer this assumption of PDF model is to reality, the closer the Bayes classifier approaches the minimum average loss in classification.

3.3 Neural Networks

The approaches discussed in the preceding subsection are based on the use of sample patterns to estimate statistical parameters of each pattern class. The minimum distance classifier needs the mean vector and the optimum one needs even the covariance matrix and a probability model**,** which directly affects the performance of the classifier. And here we want to replace all the probability assumption by training. We use these networks as vehicles for adaptively developing the coefficients of decision functions via successive presentations of training sets of patterns.

3.3.1 Two linear-separable pattern classes

We start from the simple two pattern class and assume they are linearly separable. Fig 3.1 shows schematically the perceptron (a learning unit) model for this case, and the response of this basic device is based on a weighted sum of its inputs:

$$d(\boldsymbol{x}) = \sum_{i=1}^{n} w_i x_i + w_{n+1} \tag{3.12}$$

This is a linear decision function with each components of a vector multiplying by a weight w_i, $i = 1,2,\ldots\ldots n, n+1$. The function mapping the output of the summing junction into the final output of the device is called the activation function.

The coefficient w_i, for $1 \leq i \leq n$, establish the orientation of the hyperplane, while the one with $i = n + 1$ is called the offset from the origin. To be written efficiently, an input vector $\boldsymbol{x}$ can be extended into $\boldsymbol{y}$ with the formula $\boldsymbol{y}$=$[\boldsymbol{x}^T, 1]^T$, and Eq. (3.12) becomes:

$$d(\boldsymbol{y}) = \boldsymbol{w}^{\mathrm{T}} \tag{3.13}$$

where now $\boldsymbol{w}$ is the weight vector. After setting the structure, the issue how to find $\boldsymbol{w}$ with a given training set is need solved.

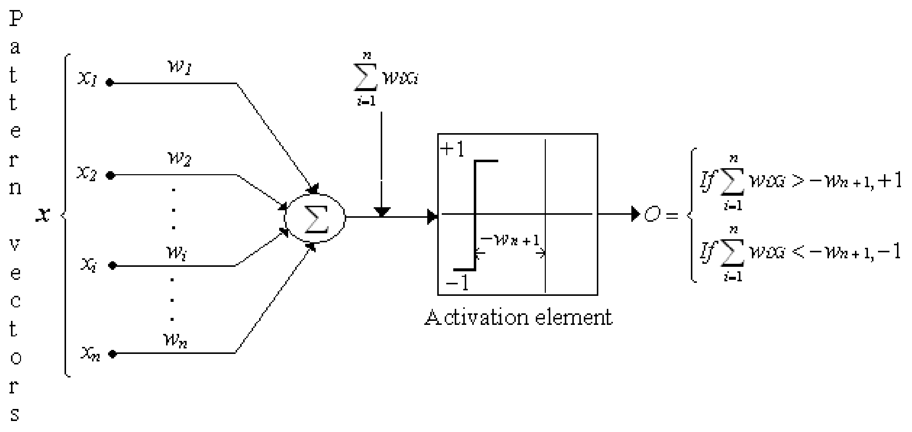


Figure 3.1: The representation of the perceptron model for two pattern classes.

Training algorithms play important roles in neural networks, which are the mechanisms used to adjust $\boldsymbol{w}$ to the best place. A simple iterative algorithm for obtaining a solution weight vector for two linearly separable training sets follows, for the condition $\boldsymbol{w}^{\mathrm{T}}(k)\boldsymbol{y}(k) \leq 0$ means pattern class w_2 and $\boldsymbol{w}^{\mathrm{T}}(k)\boldsymbol{y}(k) \geq 0$ means class w_2, at the k-th iteration step, if $\boldsymbol{y}(k) \in w_1$ while $\boldsymbol{w}^{\mathrm{T}}(k)\boldsymbol{y}(k) \leq 0$, replace $\boldsymbol{w}(k)$ by:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + c\boldsymbol{y}(k) \tag{3.14}$$

where c is a positive correction increment. This function is effective because $\boldsymbol{y}^{\mathrm{T}}(k)\boldsymbol{y}(k)$ is always nonnegative. Conversely, if $\boldsymbol{y}(k) \in w_2$ and $\boldsymbol{w}^{\mathrm{T}}(k)\boldsymbol{y}(k) \geq 0$, then replace $\boldsymbol{w}(k)$ by:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - c\boldsymbol{y}(k) \tag{3.15}$$

Otherwise, leave $\boldsymbol{w}(k)$ unchanged:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) \tag{3.16}$$

This algorithm makes a change in $\boldsymbol{w}$ only if the pattern being considered at the k-th step in the training sequences is misclassified. This algorithm is called the fixed increment correction rule. Convergence of the algorithm occurs when the entire training set for both classes is cycled through the machine without any error, and this condition exists for linear separable classes. A proof of this result is called the perceptron training theorem.

3.3.2 Two nonseparable pattern classes

Here the term, non-separable, means non-linear separable since we use a linear decision function in this section. The original delta rule, known as the Widrow-Hoff, or least-mean –square (LMS) delta rule for training perceptrons, minimizes the error between the actual and desired response at any training step:

$$J(\boldsymbol{w}) = \frac{1}{2}(r - \boldsymbol{w}^T\boldsymbol{y})^2 \tag{3.17}$$

This is the criterion and r is the desired response (+1 for w_1 and -1 for w_2 in this case). Now we want to reach the minimum of $J(\boldsymbol{w})$ by adjusting $\boldsymbol{w}$:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \alpha[\frac{\partial J\,(\boldsymbol{w})}{\partial \boldsymbol{w}}]_{\boldsymbol{w}=\boldsymbol{w}(k)} \tag{3.18}$$

where $\alpha > 0$ gives the magnitude of the correction. Following equations shows the adjusting steps:

$$\frac{\partial J\,(\boldsymbol{w})}{\partial w} = -(r - \boldsymbol{w}^T\boldsymbol{y})\boldsymbol{y} \tag{3.19}$$

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \alpha(r(k) - \boldsymbol{w}^T(k)\boldsymbol{y}(k))\boldsymbol{y}(k) \tag{3.20}$$

$$\Delta\boldsymbol{w} = \boldsymbol{w}(k+1) - \boldsymbol{w}(k) \tag{3.21}$$

$$\textbf{\textit{delta correction algorithm:}}\ \Delta\boldsymbol{w} = \alpha e(k)\boldsymbol{y}(k) \tag{3.22}$$

$$e(k) = r(k) - \boldsymbol{w}^T(k)\boldsymbol{y}(k) \tag{3.23}$$

Then if we substitute $\boldsymbol{w}(k+1)$ for $\boldsymbol{w}(k)$ in Eq. (3.23):

$$e(k) = r(k) - \boldsymbol{w}^T(k)\,\boldsymbol{y}(k) \tag{3.24}$$

$$e(k) = r(k) - \boldsymbol{w}^T(k+1)\boldsymbol{y}(k) \tag{3.25}$$

$$\begin{aligned} \Delta e(k) &= [r(k) - \boldsymbol{w}^T(k+1)] - [r(k) - \boldsymbol{w}^T(k)] \\ &= -[\mathbf{w}^{\mathrm{T}}(\mathrm{k}+1) - \mathbf{w}^{\mathrm{T}}(\mathrm{k})]\boldsymbol{y}(k) = -\Delta\boldsymbol{w}^T(k)\boldsymbol{y}(k) \\ &= -\alpha e(k)\mathbf{y}^{\mathrm{T}}(k)\boldsymbol{y}(k) = -\alpha e(k) \parallel \boldsymbol{y}(k) \parallel^2 \end{aligned} \tag{3.26}$$

Hence changing the weight reduces the error by a factor $\alpha \parallel \mathrm{y(k)} \parallel^2$. The choice of α controls stability and speed of convergence (big α has higher speed but may go outside the convergent region). There is an interesting point during the derivation of this algorithm, at first we decide to change $\boldsymbol{w}$, but the amount of change is based on $\boldsymbol{y}$, so finally we found that to change or not is entirely according to $\boldsymbol{y}$, r, and the instantaneous $\boldsymbol{w}$.

This algorithm is proved to converge to a solution that minimizes the mean square error over the patterns of the training set, while this is not intuitive for me. In the procedure, we only consider the condition for making the smallest error for a specific $\boldsymbol{y}$ rather than all the training vectors, and the best $\boldsymbol{w}$ for a certain $\boldsymbol{y}$ may not be the best $\boldsymbol{w}$ for the entire performance, there must be a average or threshold term inside the formula.

A mean-square-error solution does not imply a solution in the sense of the perceptron training theorem, which means for linear separable classes, the solution may not produce a separation hyperplane. I think it's because the choice of "initial $\boldsymbol{w}$" that results in the difference between these two algorithms. In the LMS algorithm, if the initial point is not selected well, the result may converge to a local minimum solution not a global minimum point or even no solution.

3.3.3 Multilayer feed-forward neural networks

The previous two subsections give a warm-up of neural networks, and now will start to talk about the multilayer case. Fig 3.2 shows the basic architecture and fig 3.3 shows the structure of a neuron. The number of neurons (perceptrons) in the first layer *A* is N_A and often equal to *n*, the dimension of the input pattern vector. The output layer *Q* often has N_A neurons equal to *W*, the number of pattern classes. And here we also change the hard-limiting activation function to a soft-limiting function. The following sigmoid activation function has the necessary differentiability:

$$h_j(I_j) = \frac{1}{1+e^{-(I_j+\theta_j)/\theta_0}} \tag{3.27}$$

Here, the I_j is the vector inner-product of $\boldsymbol{w}$ and the input vector of a certain layer, and θ_j is seems like w_{n+1}, while θ_0 is an offset depending on the algorithm and the purpose.

For a layer ***I*** with its preceding layer ***K***, the output of the *i*-th node of ***I*** will be:

$$O_i = h_i(I_j) = h_i\left(\sum_{k=1}^{N_K} w_{ik}\, O_k\right) = h_i\left[\sum_{k=1}^{N_K} w_{ik}\, h_i(I_k)\right] \tag{3.28}$$

A multilayer structure has two kinds of layers: output layer and hidden layer, where hidden layers mean layers other than the output layer and the name comes from their desired outputs are unknown. As a test pattern comes in, we put it into the input layer of this multilayer neural network, and we define it into a class w_j if the value of the corresponding output node is high (for example with value near to 1) and other output nodes are low (for example near to 0). We only need to define the desired output of the output layer and leave the desired output of hidden layers undefined.

Training algorithms for these two layers are different, and the training process always starts from the output layer and feed its modification back to preceding layers in each iteration. For the output layer, we still use the minimum square value criterion:

$$E_Q = \frac{1}{2}\sum_{q=1}^{N_Q}(r_q - O_q)^2 \tag{3.29}$$

where r_q is the desired result for output q and O_q is the actual result. The training algorithm is simplified as below:

$$\Delta w_{qp} = -\alpha \frac{\partial E_Q}{\partial w_{qp}} \tag{3.30}$$

$$\frac{\partial E_Q}{\partial w_{qp}} = \frac{\partial E_Q}{\partial I_q}\frac{\partial I_q}{\partial w_{qp}} \tag{3.31}$$

$$\frac{\partial I_q}{\partial w_{qp}} = \frac{\partial}{\partial w_{qp}}\sum_{p=1}^{N_P} w_{qp}\, O_p = O_p \tag{3.32}$$

$$\Delta w_{qp} = \alpha \delta_q O_p \tag{3.33}$$

The difference of algorithm between two kinds of layers is the equation for δ_q, here we generalize the definition of layer *q* and layer *p* where *q* is any layer in the structure and *p* is its preceding layer. Then the equation to achieve δ_q is as follows:

$$\delta_q = (r_q - O_q)h_q{}'(I_q) \qquad \text{for the } \textbf{\textit{output layer}} \tag{3.34}$$

$$\delta_p = h_p{}'(I_p)\sum_{q=1}^{N_q}\delta_q\, w_{qp} \qquad \text{for } \textbf{\textit{hidden layers}} \tag{3.35}$$

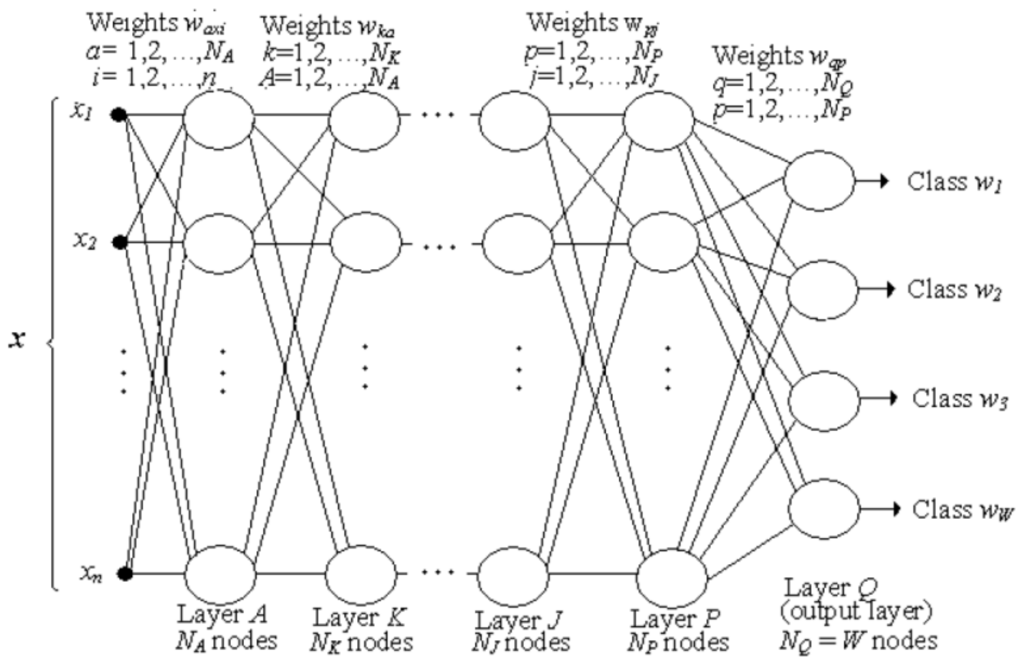


Figure 3.2: Multilayer feed-forward neural network model

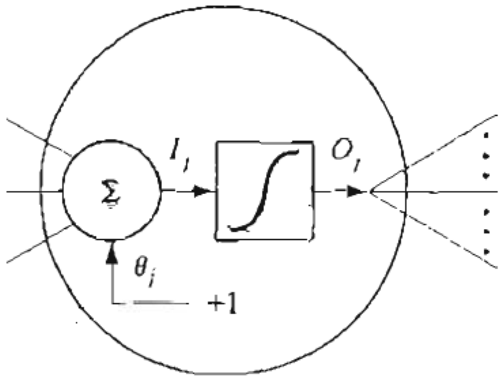


Figure 3.3: The structure of a neuron

The training step is listed below:

1. Initialization

   Assigning an arbitrary set of weights throughout the network (not equally)

2. Iterative step

   a. Computing O_q for each node of the output layer by using training vector, then generating the error terms δ_q.

   b. Appropriate error signal is passed backward to each node of the preceding layers and the corresponding weight changes are made.

3. Stopping mechanism

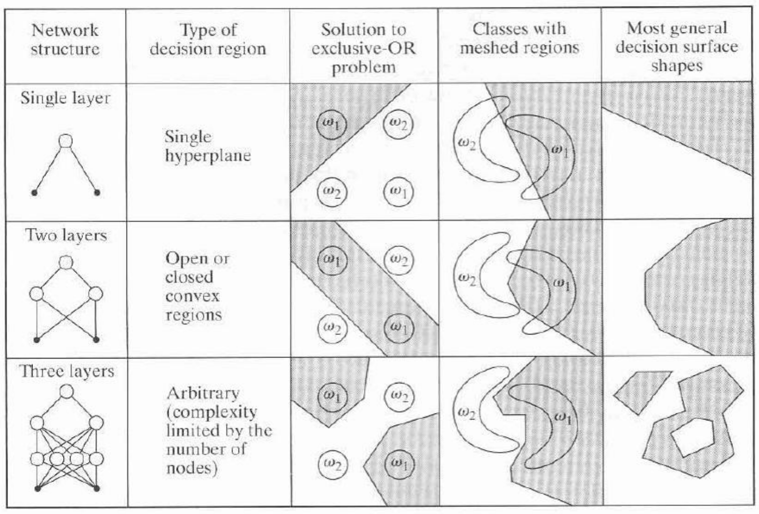
   The network error decreases with the number of iterations and the procedure converges to a stable set of weights that exhibit only small fluctuations with additional training

3.3.4 Complexity of Decision Surface

From subsections above, we know that the computation at each node is purely a linear decision function followed by an activation function, so we can see the result of each node as a linear discrimination. Take a three layer system as an instance, the effect of each layer: (assume 2 components of pattern vectors and 2 classes)

- Layer 1: line
- Layer 2: line intersection
- Layer 3: region intersection

There is a trade-off analysis between the number of layers and the number of nodes in each layer. And a three-layer network can implement decision surface of arbitrary complexity. Fig 3.4 shows examples of the complexity of decision surface.



Network structure	Type of decision region	Solution to exclusive-OR problem	Classes with meshed regions	Most general decision surface shapes
Single layer	Single hyperplane			
Two layers	Open or closed convex regions			
Three layers	Arbitrary (complexity limited by the number of nodes)			

Fig 3.4: Decision surfaces complexity implemented by multilayer networks

3.3.5 Basic setting for building a neural network

There are some terms that we have to define before a system classifier training process:

- Set the criterion for finding the best classifier
- Set the desired output
- Set the adapting mechanism

3.4 Some popular classification methods nowadays

After introducing the basic idea of the previous work of pattern recognition, we may wonder what are the most popular classifiers based on vector features nowadays. Here I'd like to show some three popular methods:

- Boosting:

  Readers who are interested in this method could find introductions in [5, 7].
- Gaussian Mixture model:

  From [4], the Gaussian mixture model is used to build the likelihood probability (conditional PDF) of the Bayesian decision function for multimodal features.
- Support vector machine:

  Readers who are interested in this method could find introductions in [7].

4. Classification methods II – Template Matching

From the preceding section, we have seen kinds of classification methods. The common point among those methods is that the pattern to be recognized is described as a vector, and the task of major concern is to appoint an unknown pattern to one of the possible classes. In this section, we assume that a set of reference (templates) are available to us and we have to decide which one of these reference patterns an unknown one (the test pattern) matches best. A reasonable first step to approaching such a task is to define a measure or a cost measuring the "distance" or the "similarity" between the (known) reference patterns and the (unknown) test pattern, in order to perform the matching operation known as template matching.

It's not that easy to select a distance (or similarity) metric as Euclidean norms and compute the distance between the template and the test pattern, there still more things we have to consider. For example, the length of two patterns may be different, and sometimes there exist orders between elements of a patter, which means that we have to carefully select the method of template matching based on its application. In the rest of this section, we'll talk about some basic idea to achieve the template matching.

4.1 Measures based on optimal path searching techniques

In this subsection, the patterns are composed of strings of identified symbols or feature vector (string patterns), that is, each of the reference and test patterns is represented as a sequence (string) of measured patterns and one has to decide which reference sequence the test one matches best. The basic structure of the optimal path searching is that we form a two-dimensional grid with the elements of the two sequences as points on the respective axes. Each point of the grid (node) marks a correspondence between the respective elements of the two sequences, and a path is an order set of nodes and used as a measurement of distance or similarity. A path is described as $(i_0, j_0)\ (i_1, j_1), \dots\dots, (i_f, j_f)$ where i_n and j_n means the *n*-th element of each sequence, and the associated overall cost *D* (distance) is defined as:

$$D = \sum_{k=0}^{K-1} d(i_k, j_k) \tag{4.1}$$

where *K* is the number of nodes along the path and $d(i_k, j_k)$ measures the distance between the respective elements of the two strings. The distance of two sequences is defined as the minimum *D* over all possible paths, and it has the function to make the alignment or warping of the test string to the element of the reference string, corresponding to the best matching score.

The distance could not only comes from the node but also comes from the transition between nodes, defined as $d(i_k, j_k | i_{k-1}, j_{k-1})$. To obtain the best path, the dynamic programming algorithms based on Bellman's principle are powerful tools that we adopt to reduce the computational complexity. The Bellman's optimality principle is shown below:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f) \text{ , the optimal path} \tag{4.2}$$

If the path passes (i, j), then the Bellman's principle states that:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f) = (i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f) \tag{4.3}$$

The consequence of this principle is that once we are at (i, j) through the optimal path, then to reach (i_f, j_f) optimally we need to search only for the optimal path from (i, j) to (i_f, j_f), and the minimum distance to reach node (i_k, j_k) could be described as:

$$D_{min}\,(i_k, j_k) = min_{i_{k-1}, j_{k-1}}[D_{min}\,(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})] \tag{4.4}$$

Fig 4.1 shows a example of a searching for optimal path.

Also there are some settings to be defined before starting a template matching:

1. Local constraint: As a transition to (i_k, j_k), this constraint limit the possible nodes allowed to be in the *(k-1)-th* position of the path.
2. Global constraints: In many cases, not all nodes but only a subset of them are included in the searching procedure, defined via the so-called global constraint. The result algorithm is known as dynamic programming.
3. End point constraints: A path should depart from the defined starting point (or a region) and end at the defined ending point (or a region) to make itself a complete path (available for finding the optimal path).
4. Cost measure: This is the most important part I thought to perform the template matching. So far we have defined the structure and method for find the distance between two patterns, while the cost of each node or each transition is defined based on the application. For example, in [6], word matching and speech matching has different consideration and has different local constraint, global constraint, end point constraint and cost measure.

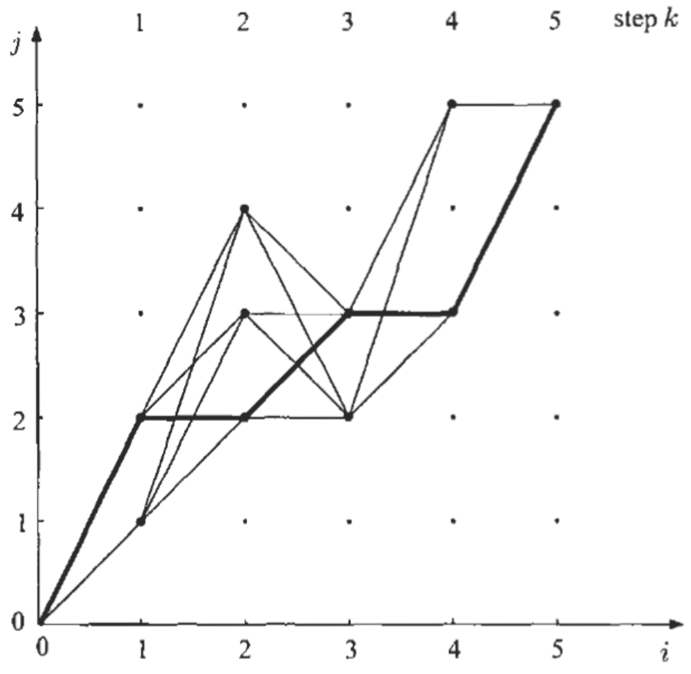


Figure 4.1: The optimal path is constructed by searching among all allowable paths, as defined by the global and local constraints. [6]

4.2 Measures based on correlation

The major task to be addressed in this subsection can be summarized as follows: "Given a block of recorded data, find whether a specific known (reference) pattern is contained within the block and where it is located". Given a test image *T* with size *I* x *J*, we want to find if a reference pattern *R* with size *M* x *N* appears or not, usually the size of *R* is smaller or equal to the size of *T*, and the task is to develop a measure for detecting an *M* x *N* sub-image in *T* that matches best the reference pattern *R*. To this end, the reference image *R* is superimposed on the test image and it is translated to all possible positions within it. There are two general ideas to compare the similarity between *R* and *T*:

1. Distance: $$D(m,n) = \sum_{i=m}^{m+M-1} \sum_{j=n}^{n+N-1} |T(i,j) - R(i-m, j-n)|^2 \quad (4.5)$$
2. Normalized correlation: $$c_N(m,n) = \frac{\sum_i \sum_j T(i,j) R(i-m, j-n)}{\sqrt{\sum_i \sum_j |T(i,j)|^2 \sum_i \sum_j |R(i,j)|^2}} \quad (4.6)$$

, where i, j means the overlap region under translation (m,n)

In fact, besides translation, the reference pattern is possible to be rotated or scaled and makes the matching task complicated. Here we may use the invariant moment to describe patterns, or use some RST-invariant transform [8] such as the Fourier-Mellin transform to get an invariant representation. Another way of thinking is to collect a set of distorted reference patterns to cover all possibilities while it needs more matching computations, and the Karhunen-Loeve transform could be use to simplify the representation of each reference pattern in a set.

4.3 Computational consideration and improvement

To perform the matching of subsection 4.2, we need to consider the computational cost. It is usually more efficient to compute the cross-correlation via its Fourier transform, and in some case we'll limit the searching range in a defined region such as the motion compensation of video coding. If we use a rectangular window $[-p,p]\times[-p,p]$ centered at a point $(x,\ y)$ in $T\ (i,\ j)$ and without using Fourier domain computation, we still require a number proportional to $(2p+1)^2MN$ additions and multiplications which leads to a huge number of operation. Thus, in practice, suboptimal heuristic searching techniques are usually adopted. There are two major directions: one is to reduce the search points and the other one is to reduce the size of the involved images.

1. Two-dimensional logarithmic search: Fig 4.2.
2. Hierarchical search: This technique springs from the multi-resolution concept, starts from the similarity search at the lowest resolution and then searches among the neighbors of the optimal point at the higher resolution layer.
3. Sequential methods: When counting the difference of two patterns, a threshold is set and the error is computed in a smaller and sequentially increasing window. So if the error exceeds the threshold before the window reaches its original size, the computation stops and starts at another translation position.

4.4 Deformable template matching

In subsection 4.2, we assume that the template and the object, residing in the image, were identical. However, there are many problems where we know a priori that the available template and the object we search for in the image may not look exactly the same. Our goal in this subsection is to allow the template matching procedure to account for deviation between the reference template and the corresponding test pattern in the image. Here we assume the reference template is available in the form of an image array containing the object's boundary information but discards texture. Figure 4.3 shows an example of object boundary matching.

A reference template array $R(i,j)$ is defined as prototype, and the basic idea behind the deformable template matching procedure is to deform the prototype and produce deformed variants of it. Some mechanisms to achieve the deformable template matching shows as below:

1. Prototype: Maybe the mean shape characteristics of the corresponding shape class.
2. A transform procedure to deform the prototype: Which means the way to modify the prototype
3. A criterion to define the best match: For each prototype, there exists a best match as the measure of the similarity to the test pattern. And the criterion to define the best match is shown as:

   $$\xi:\ \min_{\xi}\{E_m(\xi) + E_d(\xi)\} \qquad (4.7)$$

   where ξ means the deformation parameter of the prototype. The term $E_m(\xi)$ (matching energy) stands for the distance between the deformed template and the test pattern, while the $E_d(\xi)$ (deformation energy) is the cost to modify the prototype to the corresponding deformed template.

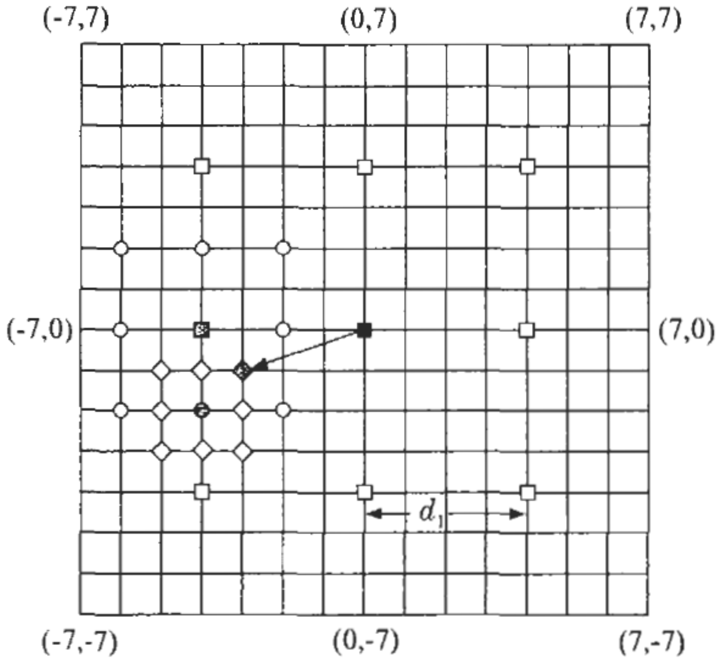
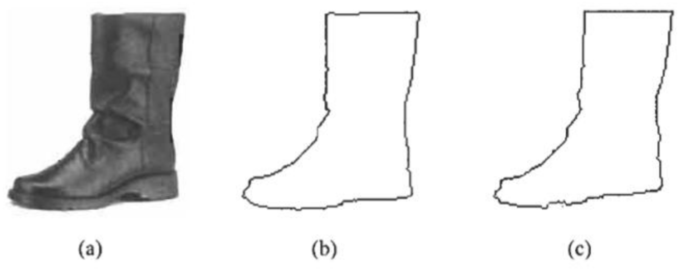


Figure 4.2: Logarithm search to find the point of maximum cross-correlation. [6]



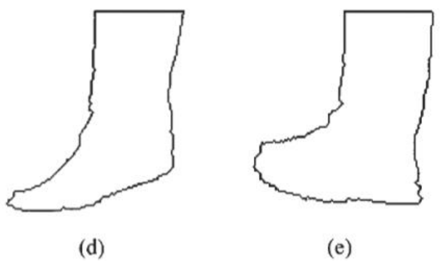


Figure 4.3: (a) A reference pattern, (b) its contour used as prototype, and (c), (d), (e) three of its deformed variants.

5. Context-dependent methods

In this section we'll talk about some recognition cases that there exist some relations between each classes. In other words, the class to which a feature vector is assigned depends (a) on its own value, (b) on the values of the other feature vectors, and (c) on the existing relation among the various classes. Here we have to consider more about the mutual information, which resides within the feature vectors, requiring the classification to be performed using all feature vectors simultaneously and also to be arranged in the same sequence in which they occurred from the experiments.

5.1 Extension of the Bayes classifier

In the preceding sections, we have learned the basic idea that based on the minimum error task, an observation $\boldsymbol{x}$ will be classified into the i-th class w_i if:

$$P(w_i|\boldsymbol{x}) > P(w_j|\boldsymbol{x}) \text{ for } i,j \in [1,M] \text{and } i \neq j \tag{5.1}$$

And now we'll extend this idea into a sequence of N observations X:$\boldsymbol{x_1}, \boldsymbol{x_2}, \dots \dots, \boldsymbol{x_N}$ and M classes: $w_1, w_2, \dots \dots, w_M$ that each observation will be classified into. Let $\Omega_i: w_{i1}, w_{i2}, \dots \dots, w_{iN}$ be one of the possible sequences of these classes corresponding to the observation sequence, with $i_k \in \{1, \dots \dots M\}$ for $k = 1,2, \dots \dots N$. Our classification task now is to decide to which class sequence Ω_i a sequence of observation X corresponds. Then now the observation sequence could be decided into the i-th class sequence based on the following formulas:

$$P(\Omega_i|\boldsymbol{X}) > P(\Omega_J|\boldsymbol{X}) \quad for\ i,j \in [1, M^N] and\ i \neq j \tag{5.2}$$

$$P(\Omega_i)P(\boldsymbol{X}|\Omega_i) > P(\Omega_J)P(\boldsymbol{X}|\Omega_J) \quad for\ i,j \in [1, M^N] and\ i \neq j \tag{5.3}$$

5.2 Markov chain model

One of the most widely used models describing the underlying class dependence is the Markov chain rule. Here we'll talk about the first-order one and two assumptions are made by this rule to simplify the task:

$$P\left(w_{i_k}|w_{i_{k-1}},w_{i_{k-2}},\dots\dots,w_{i_1}\right)=P\left(w_{i_k}|w_{i_{k-1}}\right) \tag{5.4}$$

, where *k* means the *k*-th class in the sequence

$$P\left(\boldsymbol{x_k}|w_{i_k},w_{i_{k-1}},w_{i_{k-2}},\dots\dots,w_{i_1},\boldsymbol{X}-\boldsymbol{x_k}\right)=P\left(\boldsymbol{x_k}|w_{i_k}\right) \tag{5.5}$$

Then based on these assumptions we can get the probability terms as following:

$$P(\Omega_i)=P\left(w_{i_1}\right)\prod_{k=2}^{N}P\left(w_{i_k}|w_{i_{k-1}}\right) \tag{5.6}$$

, where $P\left(w_{i_1}\right)$ is the prior probability for class w_{i_1}

$$P(\boldsymbol{X}|\Omega_i)=\prod_{k=1}^{N}P\left(\boldsymbol{x_k}|w_{i_k}\right) \tag{5.7}$$

Combining these two sequences, we can get the Bayes rule for Markovian nodels and decide the *i*-th class sequence to be decided if:

$P(\boldsymbol{X}|\Omega_i)P(\Omega_i)=P\left(w_{i_1}\right)P\left(\boldsymbol{x_1}|w_{i_1}\right)\prod_{k=2}^{N}P\left(w_{i_k}|w_{i_{k-1}}\right)P\left(\boldsymbol{x_k}|w_{i_k}\right)$ the max one (5.8)

5.3 The Viterbi Algorithm

If we directly compute the before equation, we need a total of $O(NM^N)$ multiplications, which is usually a large number indeed. While there exists a rich computational structure in eq. (5.8) and we'll exploit it in this subsection.

Fig 5.1 shows a diagram with N dot columns, where each dot in a column represents one of the M possible classes, $w_1,w_2,\dots\dots,w_N$. The arrows determine transitions form one class to another, as observation vectors are obtained in sequence. Thus, each of the class sequence Ω_i corresponds to a specific path of successive transitions. Each transition from one class w_i to w_j is characterized by a fixed probability $P\left(w_j|w_i\right)$ (assume known by modeling, independent to the stage where it occurs), and $P(\boldsymbol{x}|w_i)$ is also known during the modeling process**.** Thus, the classes along this optimal path (in the sense of the largest probability) will be the ones in which the respective observations are classified.

Here we can use the idea introduced in 4.1 to find the optimal path, some terms are shown below:

Cost function of a transition: $d\left(w_{i_k},w_{i_{k-1}}\right)=P\left(w_{i_k}|w_{i_{k-1}}\right)P\left(\boldsymbol{x_k}|w_{i_k}\right)$for k $\neq$ 1(5.9)

$d\left(w_{i_1},w_{i_o}\right)=P\left(w_{i_1}\right)P\left(\boldsymbol{x_k}|w_{i_1}\right)$ for k = 1(5.10)

The overall cost: $D\equiv\prod_{1}^{N}d\left(w_{i_k},w_{i_{k-1}}\right)$ the biggest D is the optimal one (5.11)

Take the logarithm: $\mathrm{D}=\sum_{1}^{\mathrm{N}}d\left(w_{i_k},w_{i_{k-1}}\right)$ (5.12)

Then the Bellman's principle could again offer us the means for efficient optimization, hence the amount of computation reduces to $O(NM^2)$, and the resulting algorithm is known as Viterbi algorithm.

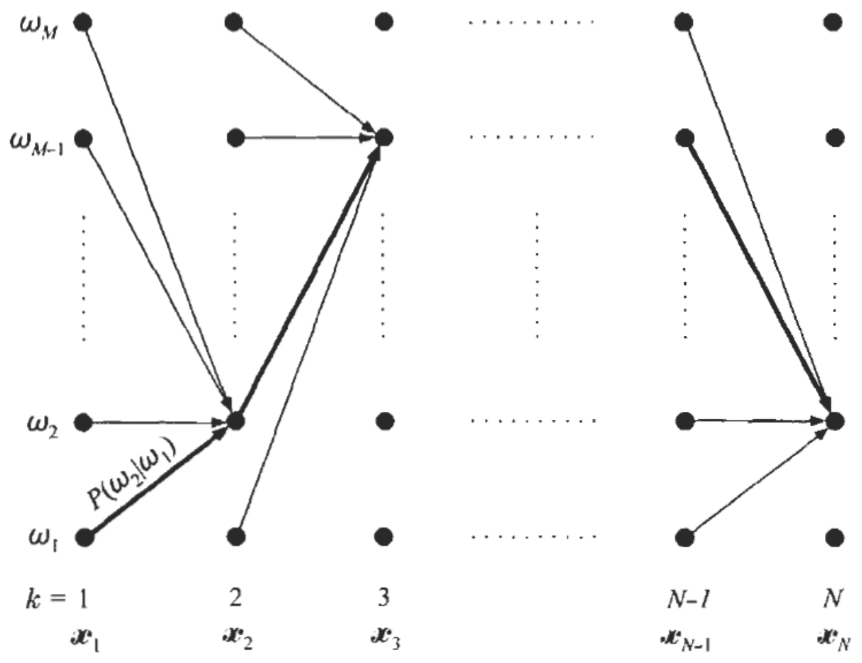


Figure 5.1: Trellis diagram of the Viterbi algorithm [6]

5.4 Hidden Markov models

In the Markov chain training phase, the state (cluster) of each observation could be recognized directly, which means during the training period these states can be labeled and we can estimate their associated parameters. While in some cases the states are not directly observed and can only be inferred from the sequence of the observations via some optimization technique. These types of Markov models are known as hidden Markov models (HMMs).

An HMM is a type of stochastic modeling appropriate for non-stationary stochastic sequence, with statistical properties that undergo distinct random transitions among a set of different stationary process. In other words, an HMM models a sequence of observations as a piecewise stationary process. The process to generate an HMM consists of setting a modeling structure, and training the parameter of it. The reason for indirect observations is because the labeling should obey the structure, which means some transitions are impossible in the corresponding HMM. Fig 5.2 shows an example of an HMM.

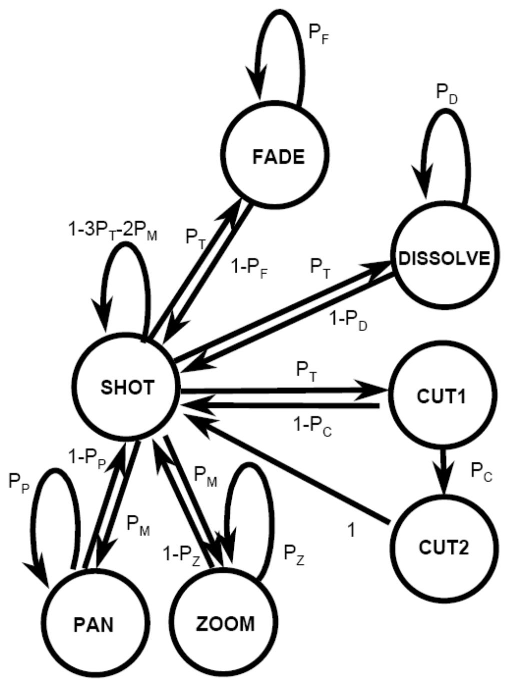


Figure 5.2: HMM for video segmentation [4]

There are two main usages of HMMs, one is that there are several classes which are modeled by HMM and we want to classify the observed sequence into one of them; another one is that a HMM is generated for a certain event such as the video segmentation in [4]. For the recognition task, the second usage could easily be executed by Viterbi algorithm, while for the first one there are two different criterions in use.

Recognition:
Assume we have:
(1)M possible classes (models)
(2)each model has K_s states ($s \in [1, M]$)
(3)The PDF $P(\boldsymbol{x}|j)$ ($j \in [1, K_s]$)
(4)The transition probability $P(i|j)$
(5)The initial probability $P(i)$ ($i \in [1, K_s]$)

We have two recognition metrics for the first case:
(1) All path method: Count the cost of all paths in a model and compare them to decide which class to be classified.
(2) Best path method: Find the optimal path of each model and compare them to

decide which class to be classified.

All path method:

Each HMM could be described as: $S = \{P(i|j), P(\boldsymbol{x}|i), P(i), K_s\}$ (5.13)

$$P(\boldsymbol{X}|S) = \sum_i P(\boldsymbol{X}, \Omega_i|S) = \sum_i P(\boldsymbol{X}|S, \Omega_i)\, P(\Omega_i|S) \quad (5.14)$$

$$\alpha(i_{k+1}) \equiv P(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots\dots, \boldsymbol{x_{k+1}}, i_{k+1}|S) = \sum_{i_k} \alpha(i_k)\, P(i_{k+1}|i_k) P(\boldsymbol{x_{k+1}}|i_{k+1}) \quad (5.15)$$

$$k = 1,2, \dots\dots, N-1 \text{ with } \alpha(i_1) = P(i_1)P(\boldsymbol{x_1}|i_1)$$

$$\beta(i_k) \equiv P(\boldsymbol{x_{k+1}}, \boldsymbol{x_{k+2}}, \dots\dots, \boldsymbol{x_N}|i_k, S) = \sum_{i_k+1} \beta(i_{k+1})\, P(i_{k+1}|i_k) P(\boldsymbol{x_{k+1}}|i_{k+1}) \quad (5.16)$$

$$\beta(i_N) = 1, i_N \in [1,2, \dots\dots, K_s]$$

$$\gamma(i_k) \equiv P(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots\dots, \boldsymbol{x_N}, i_k|S) = \alpha(i_k)\, \beta(i_k) \quad (5.17)$$

And the $P(\boldsymbol{X}|S)$ we want for the all path method is defined as:

$$P(\boldsymbol{X}|S) = \sum_{i_N=1}^{K_s} \alpha(i_N) \quad (5.18)$$

Best path method:

Just following the Viterbi algorithm

The most difficult part may be the training process because the observations are indirect.

Training algorithm:

(1) Baum-Welch re-estimation: Used for the all path method

(2) Viterbi re-estimation: Used for the optimal path method

Another thing we have to consider is the form of observation $\boldsymbol{x}$, if it is pre-quantized in a discrete type, then the conditional probability term $P(\boldsymbol{x}|j)$ can be stored in a "look-up table" form. While for the $\boldsymbol{x}$ described in a continuous form, a probability model is needed, and the suitable probability model is the "Mixture model" [6].

6. Feature Generation

This chapter will discuss how to generate features for a pattern, and the more pertinently these features describe the pattern, the higher probability the classifier could recognize the pattern accurately. In this report, this task is focused on image analysis. Our major goal here is that given an image, or a region within an image, generating the features that will subsequently be fed to a classifier in order to classify the image in one of the possible classes.

The need for feature generation stems from our inability to use the raw data, one reason is that the raw data is too big to deal with, and the other reason is that the raw data can't give the classifier the same sense what people feel about the image. As a result, feature generation is aimed to exhibit high information packing properties, from the class separability point of view. The rest of this chapter is arranged base on the time each technique proposed.

6.1 Regional feature

In this subsection we try methods to describe the texture, the distribution of color (or gray level) and the relation between each pixel.

6.1.1 Features for texture characterization

The texture of an image region is determined by the way the gray levels are distributed over pixels in this region. Even there is no clear definition of "texture", we are all in a position to describe an image by the look of it as fine or coarse, smooth or irregular, homogeneous or inhomogeneous.

First-order statistical features:

Based on the histogram $H(I)$ of a region of interest, where I is the possible intensity of a gray-level image and N_g is the number of possible gray levels, kinds of moment can be defined as following:

Moment: $$m_i = E\left[I^i\right] = \sum_{I=0}^{N_g-1} I^i\, P(I) \qquad i = 1,2,\ldots \qquad (6.1)$$

Central moment: $$\mu_i = E\left[(I - E[I])^i\right] = \sum_{I=0}^{N_g-1} (I - m_1)^i\, P(I) \qquad (6.2)$$

The most frequently used are m_1, μ_2(variance), μ_3(skewness), and μ_4(kurtosis) of the histogram. Fig 6.1 shows six variations of the same image (with 16 gray levels) with their corresponding histogram, and the resulting $\mu_3$and μ_4 from left to right are listed in table 6.1. Other quantities resulting from the first-order histogram are:

Absolute moments: $$\hat{\mu}_i = E\left[|I - E[I]|^i\right] = \sum_{I=0}^{N_g-1} |I - m_1|^i\, P(I) \qquad (6.3)$$

Entropy: $$H = -E[log_2 P(I)] = -\sum_{I=0}^{N_g-1} P(I) \log_2 P(I) \qquad (6.4)$$

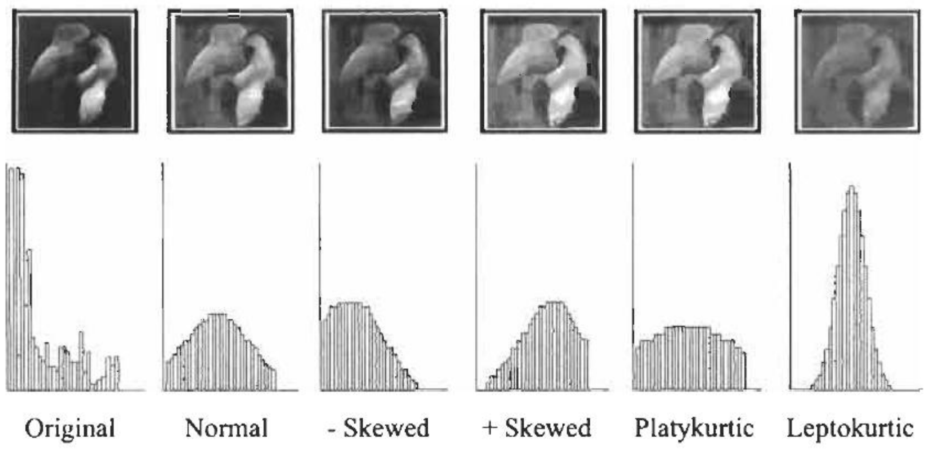


Figure 6.1: Examples of images and corresponding histograms [6]

Table 6.1: Table for $\mu_3$and μ_4 from left to right of fig 6.1.

μ_3:	587	0	-169	169	0	0
μ_4:	16609	7365	7450	7450	9774	1007

Second-order statistical features—co-occurrence matrices

Here the relative positions of the various gray levels within the image are taken into consideration which means now we consider a pair of pixels together, and two more parameters (relative distance among the pixels and their relative orientation) enter into the scene. Let *d* be the relative distance and the orientation ϕ be quantized in four directions: $(0^o, 45^o, 90^o, 135^o)$, then for each combination of d and ϕ a 2-D histogram is defined:

$$0^o: P(I(m,n) = I_1, I(m \pm d, n) = I_2) \text{ as an example} \tag{6.5}$$

And for each of these histograms an array is defined, known as the co-occurrence or spatial dependence matrix. Based on this matrix, several moments can be defined:

Angular second moment (smoothness): $ASM = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} (P(i,j))^2$ (6.6)

Contrast: $CON = \sum_{i=0}^{N_g-1} n^2 \left\{\sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} P(i,j)\right\} \; with\; |i-j| = n$ (6.7)

Inverse difference moment: $IDF = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} \frac{P(i,j)}{1+(i-j)^2}$ (6.8)

Entropy: $H_{xy} = -\sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} P(i,j)\, log_2 P(i,j)$ (6.9)

6.1.2 Local linear transforms for texture extraction

This method aimed at using the linear transform to extract the texture information, and the most important task is how to define the transfer matrix. For example, transform methods such as DCT, DST, KLT, and Gabor have been used.

6.1.3 Geometric moments

Here we take the position of pixel and its pixel value together into account. With some modification, these moments can be rotation, translation, and scaling invariant. A powerful moment, Zernike moments has the properties that the basis of each moment is orthogonal to each other. Readers who are interested in the geometric moments could find more information in [8].

6.1.4 Parametric models

This method comes from the concept of random process, where we define a model of generation and estimate the corresponding parameter which can approximate the pattern best (means least error). The well-know AR model is a good example of parametric models. If the model is defined pertinent, then this method could dramatically reduce the description (only these parameters) of patterns and still maintain good quality. While this method is application-dependent and if the model is not well-defined, the description will be far away from the exact pattern.

6.2 Feature for shape and size characterization

For some pattern recognition case, we focus on the shape and size of objects not the texture, such as the automatic character recognition in an optical character recognition (OCR) system. A widely-used feature here is the boundary curve, and before the description process of the boundary, we need some preprocessing to find the boundary. The general process for finding boundary consists of segmentation algorithm, binarization, and boundary extraction, and fig 6.2 shows the result of each step.

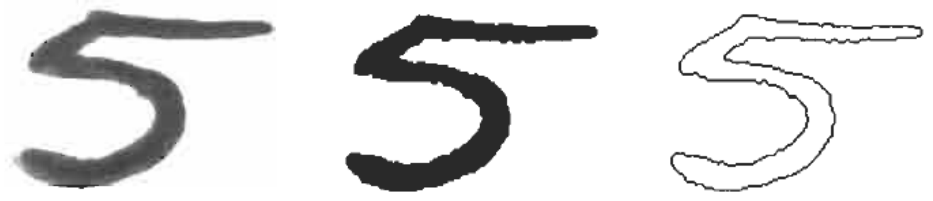


Figure 6.2: The character “5” after (a) the segmentation of the scanned image and then (b) the application of a binarization algorithm and (c) its boundary after the application of a boundary extraction algorithm in the binarization version.

What is of paramount importance in such system is feature invariance in geometric transformation. The recognition of the character must be insensitive to its position, size, and orientation. There are two major directions to describe the boundary, one is the invertible transform such as Fourier transform, and another one is to use features that are descriptive of the characteristics of the shape of the region but are not regenerative. Examples of such features are the corners in the boundary and the perimeter.

6.2.1 Fourier transform

The boundary pixel position (x_k, y_k) could be represented as $u_k = x_k + y_k$, and the N u_k points could used to obtain the N-point DFT:

$$f_l = \sum_{k=0}^{N-1} u_k \, exp(-j\frac{2\pi}{N}lk), \qquad l = 0,\ 1, \ldots\ldots,\ N\text{-}1 \tag{6.10}$$

There are some properties of Fourier transform makes it RST invariant [6]. And we can also change the input sequence form of eq. (6.10) to reach another result of Fourier transform.

6.2.2 Chain Codes

Chain coding is among the most widely used techniques for boundary shape description [9]. The boundary is approximated via a sequence of connected straight line segments of preselected direction and length. Every line segment is coded with a specific coding number depending on its direction (usually 4-directional or 8-directional). The disadvantage of this description is that the resulting chain codes are usually long and at the same time are very sensitive in the presence of noise, and some pre-processing such as resampleing the boundary curve by selecting a grid of larger dimensions has been proposed to solve this problem. Fig 6.3 shows the resampling process.

After the processing above, now we can get a better chain code. Also there exist different methods to describe the boundary, such as record the difference of angle between adjacent edges, and finally we can save this description as a histogram or as a string.

6.2.3 Moment-based features

Here we can also use the same method, geometric moments, described in 6.1.3 to represent only the boundary pixels.

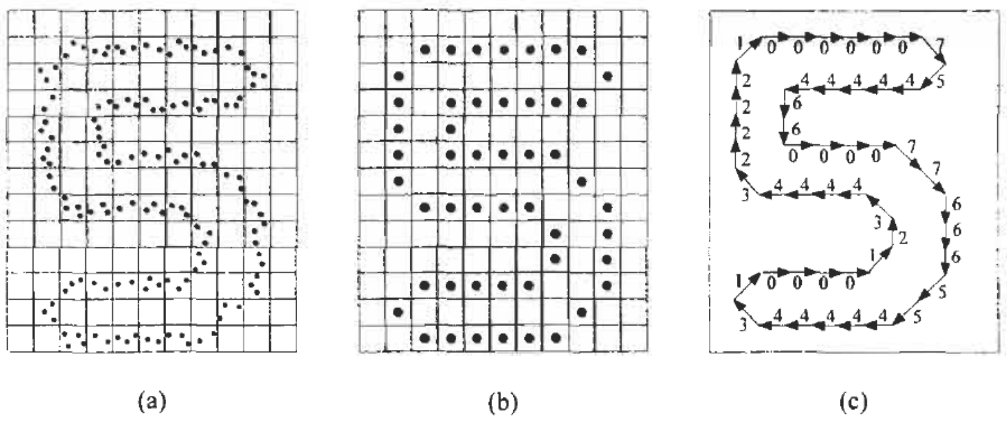


Figure 6.3: The character “5” and (a) its original sampled image, (b) its resampled version on a coarser grid, and (c) the resulting chain code.

6. Conclusion

Pattern recognition is nearly everywhere in our life, each case relevant to decision, detection, retrieval can be a research topic of pattern recognition. The mathematics of pattern recognition is widely-inclusive, the methods of game theory, random process, decision and detection, or even machine learning can used for this task, which make pattern recognition a really wide research.

From the studies these eight weeks, I have overviewed kinds of pattern recognition methods, some are the early methods and some are the newly methods. The Bayes theorem, neural network gives a basic structure of classifier, and later these classifiers are used for different cases with different modifications. The modeling concepts give a different way of pattern recognition, which analyze the observation by its synthesis process and can reduce a large amount of data of the features, while it suffers from strong dependence of applications and hard to be generalized.

Towards research of pattern recognition in the future, I found there are two directions. One is to decide a case, and try to use different features with different classifiers to see if we can recognize this case, for example, the face recognition. And another direction is to research the structure of classifier or the relation between pattern recognition and information theory, which could gives a theoretical concepts of pattern recognition. For example, we can estimate the effectiveness of classification or features under a specific case assumption. In the chapter 9 of [6], the channel equalization case gives me a notice that the communication problem could also be solved by pattern recognition.

Also in the recent work, we want not only to generate a system for recognition but also give it the ability of learning and adaption. In my opinion, machine learning and pattern recognition complement to each other, which means the concepts of pattern recognition could be used for design a proper learning algorithm, while the learning algorithm could be used to enhance the result of pattern recognition.

There still much I have to learn for this big filed of research, and in the future if I continue my research on this topic, I'd like to learn more about its relevance between machine learning and information theory.

Reference

[1] R. C. Gonzalez, "Object Recognition," in *Digital image processing*, 3rd ed. Pearson, August 2008, pp. 861-909.

[2] Ke-Jie Liao, *"Image-based Pattern Recognition Principles,"* August 2008. [online] Available: http://disp.ee.ntu.edu.tw/research.php. [Accessed Sep. 19, 2009].

[3] Shyh-Kang Jeng, "Introduction", Pattern recognition Course Website, 2009. [online] Available: http://cc.ee.ntu.edu.tw/~skjeng/PatternRecognition2007.htm. [Accessed Sep. 30, 2009].

[4] J.S. Boreczky, L.D. Wilcox, "A hidden Markov model framework for video segmentation using audio and image features," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP-98)*, Vol. 6, Seattle, WA, May 1998.

[5] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. John Wiley & Sons, 2001.

[6] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, 2nd ed. Academic Press, 2003.

[7] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2004.

[8] J. Wood, "Invariant pattern recognition," *Pattern Recognition*, Vol. 29(1), pp. 1-17, 1996.

[9] Freeman H. "On the encoding of arbitrary geometric configurations," *IRE Transactions* on *Electronic Computers*, Vol. 10(2), pp. 260-268, 1961.