SGN-2506: Introduction to Pattern Recognition

Jussi Tohka
Tampere University of Technology
Department of Signal Processing

2006 - 2008

March 12, 2013

Preface

This is an English translation of the lecture notes written originally in Finnish for the course SGN-2500 *Johdatus hahmontunnistukseen.* The basis for the original lecture notes was the course *Introduction to Pattern Recognition* that I lectured at the Tampere University of Technology during 2003 and 2004. Especially, the course in the fall semester of 2003 was based on the book *Pattern Classification, 2nd Edition* by Richard Duda, Peter Hart and David Stork. The course has thereafter diverted from the book, but still the order of topics during the course is much the same as in the book by Duda et al.

These lecture notes correspond to a four credit point course, which has typically been lectured during 24 lecture-hours. The driving idea behind these lecture notes is that the basic material is presented thoroughly. Some additonal information are then presented in a more relaxed manner, for example, if the formal treatment of a topic would require too much mathematical background. The aim is to provide a student with a basic understanding of the foundations of the statistical pattern recognition and a basis for advanced pattern recognition courses. This necessarily means that the weight is on the probabilistic foundations of pattern recognition, and specific pattern recognition techniques and applications get less attention. Obviously, introducing specific applications could be motivating for the student, but because of the time limitations there are few chances for this. Moreover, understanding of the foundations of the statistical pattern recognition is necessary to grasp specific techniques used in an application of interest.

I have made no major modifications to these lecture notes since 2006 besides correcting some obvious errors. Discussions with Jari Niemi and Mikko Parviainen have helped to shape some of the material easier to read.

Tampere fall 2008

Jussi Tohka

Contents

Chapter 1

Introduction

The term *pattern recognition* refers to the task of placing some object to a correct class based on the measurements about the object. Usually this task is to be performed automatically with the help of computer. Objects to be recognized, measurements about the objects, and possible classes can be almost anything in the world. For this reason, there are very different pattern recognition tasks. A system that makes measurements about certain objects and thereafter classifies these objects is called a *pattern recognition system.* For example, a bottle recycling machine is a pattern recognition system. The customer inputs his/her bottles (and cans) into the machine, the machine recognizes the bottles, delivers them in proper containers, computes the amount of compensation for the customer and prints a receipt for the customer. A spam (junk-mail) filter is another example of pattern recognition systems. A spam filter recognizes automatically junk e-mails and places them in a different folder (e.g. /dev/null) than the user's inbox. The list of pattern recognition systems is almost endless. Pattern recognition has a number of applications ranging from medicine to speech recognition.

Some pattern recognition tasks are everyday tasks (e.g. speech recognition) and some pattern recognition tasks are not-so-everyday tasks. However, although some of these task seem trivial for humans, it does not necessarily imply that the related pattern recognition problems would be easy. For example, it is very difficult to 'teach' a computer to read hand-written text. A part of the challenge follow because a letter 'A' written by a person B can look highly different than a letter 'A' written by another person. For this reason, it is worthwhile to model the variation within a class of objects (e.g. hand-written 'A's). For the modeling of the variation during this course, we shall concentrate on statistical pattern recognition, in which the classes and objects within the classes are modeled statistically. For the purposes of this course, we can further divide statistical pattern recognition into two subclasses. Roughly speaking, in one we model the variation within object classes (generative modeling) and in the other we model the variation between the object classes (discriminative modeling). If understood broadly, statistical pattern recognition covers a major part of all pattern recognition applications and systems.

Syntactic pattern recognition forms another class of pattern recognition methods. During this course, we do not cover syntactic pattern recognition. The basic idea of syntactic pattern recognition is that the patterns (observations about the objects to be classified) can always be represented with the help of simpler and simpler subpatterns leading eventually to atomic patterns which cannot anymore be decomposed into subpatterns. Pattern recognition is then the study of atomic patterns and the language between relations of these atomic patterns. The theory of formal languages forms the basis of syntactic pattern recognition.

Some scholars distinguish yet another type of pattern recognition: The neural pattern recognition, which utilizes artificial neural networks to solve pattern recognition problems. However, artificial neural networks can well be included within the framework of statistical pattern recognition. Artificial neural networks are covered during the courses 'Pattern recognition' and 'Neural computation'.

Chapter 2

Pattern Recognition Systems

2.1 Examples

2.1.1 Optical Character Recognition (OCR)

Optical character recognition (OCR) means recognition of alpha-numeric letters based on image-input. The difficulty of the OCR problems depends on whether characters to be recognized are hand-written or written out by a machine (printer). The difficulty level of an OCR problem is additionally influenced by the quality of image input, if it can be assumed that the characters are within the boxes reserved for them as in machine readable forms, and how many different characters the system needs to recognize.

During this course, we will briefly consider the recognition of hand-written numerals. The data is freely available from the machine learning database of the University of California, Irvine. The data has been provided by E. Alpaydin and C. Kaynak from the University of Istanbul, see examples in Figure 2.1.

2.1.2 Irises of Fisher/Anderson

Perhaps the best known material used in the comparisons of pattern classifiers is the so-called Iris dataset. This dataset was collected by Edgar Anderson in 1935. It contains sepal width, sepal length, petal width and petal length measurements from 150 irises belonging to three different sub-species. R.A. Fisher made the material famous by using it as an example in the 1936 paper 'The use of multiple measurements in taxonomic problems', which can be considered as the first article about the pattern recognition. This is why the material is usually referred as the Fisher's Iris Data instead of the Anderson's Iris Data. The petal widths and lengths are shown in Figure 2.2.

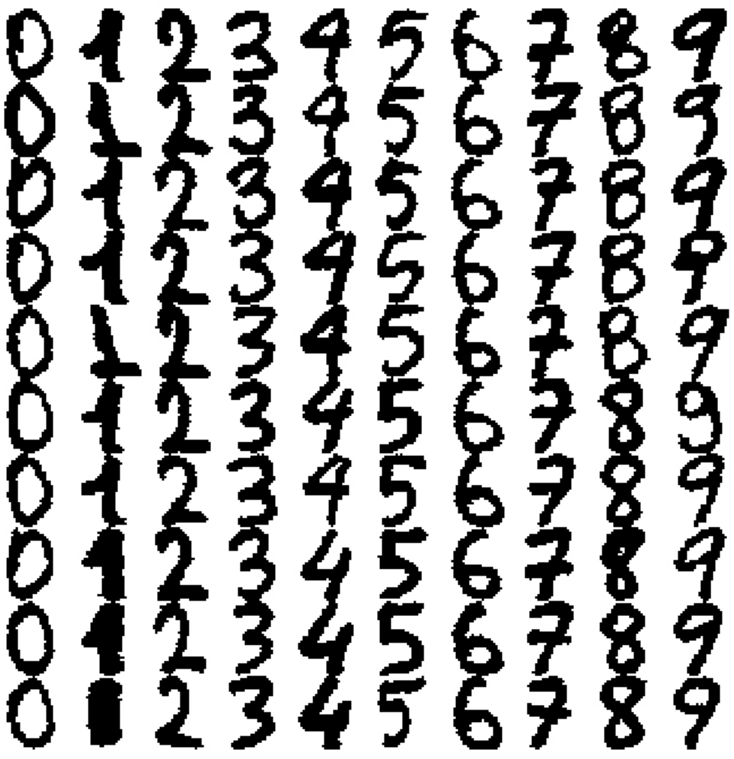

Figure 2.1: Handwritten digits, 10 examples of each digit. All digits represented as 32×32 binary (black and white) image.

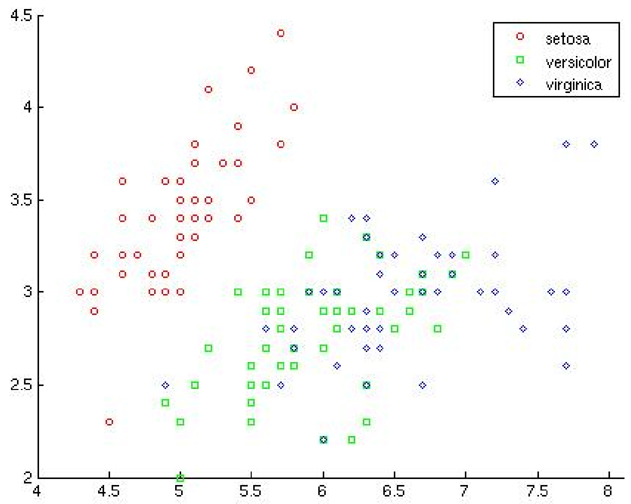

Figure 2.2: Petal widths (x-axis) and lengths (y-axis) of Fisher's irises.

2.2 Basic Structure of Pattern Recognition Systems

The task of the pattern recognition system is to classify an object into a correct class based on the measurements about the object. Note that possible classes are usually well-defined already before the design of the pattern recognition system. Many pattern recognition systems can be thought to consist of five stages:

1. Sensing (measurement);
2. Pre-processing and segmentation;
3. Feature extraction;
4. Classification;
5. Post-processing;

A majority of these stages are very application specific. Take the sensing stage as an example: the measurements required by speech recognition and finger print identification are rather different as are those required by OCR and Anderson's Iris measurements.

For the classification stage, the situation is different. There exist a rather general theory about the classification and an array of general algorithms can be formulated for the classification stage. For this reason, the main emphasis during this course is on the classifier design. However, the final aim is to design a pattern recognition system. Therefore it is important to gain understanding about every stage of the pattern recognition system although we can give few application non-specific tools for the design of these stages in a pattern recognition system.

Sensing refers to some measurement or observation about the object to be classified. For example, the data can consist of sounds or images and sensing equipment can be a microphone array or a camera. Often one measurement (e.g. image) includes information about several objects to be classified. For instance, assume that we want to recognize the address written on the envelope. We must then classify several characters to recognize the whole address. The data here is probably an image of the envelope including the sub-images of all the characters to be classified and some background that has nothing to do with the pattern recognition task.

Pre-processing refers to filtering the raw data for noise suppression and other operations performed on the raw data to improve its quality. In **segmentation**, the measurement data is partitioned so that each part represents exactly one object to be classified. For example in address recognition, an image of the whole address needs to be divided to images representing just one character. The result of the segmentation can be represented as a vector that is called a *pattern vector*.

Feature extraction. Especially when dealing with pictorial information the amount of data per one object can be huge. A high resolution facial photograph (for face recognition) can contain 1024*1024 pixels. The pattern vectors have then over a million components. The most part of this data is useless for classification. In feature extraction, we are searching for the features that best characterize the data for classification The result of the feature extraction stage is called a *feature vector*. The space of all possible feature vectors is called the *feature space*. In face recognition, a widely used technique to reduce the number features is principal component analysis (PCA). (This yields so called eigenfaces.) PCA is a statistical technique to reduce the dimensionality of a data vector while retaining most of the information that the data vector contains. In addition to mathematical/computational/statistical techniques, feature extraction can be performed heuristically by picking such features from a pattern vector that could be assumed to be useful in classification. For face recognition, such a feature can be the distance between the two eyes of a person. Sometimes, dimensionality of the input data is very limited and the pattern vector can be chosen as the feature vector. For example this is the case with the image segmentation methods that are based on the pattern recognition principles. Feature extraction is highly application specific although some general techniques for feature extraction and selection have been developed. You will hear more about these during the course 'Pattern Recognition'.

In general, the line between feature extraction and classification is fuzzy. The task of the feature extractor is to produce a representation about the data that enables an easy classification. On the other hand, the task of the classifier is to produce the best classification accuracy given the extracted features. Clearly, these two stages are interdependent from the application point of view. Also, and perhaps more importantly, what is the best representation for the data depends on the classifier applied. There is no such thing as the universally optimal features.

The classifier takes as an input the feature vector extracted from the object to be classified. It places then the feature vector (i.e. the object) to class that is the most appropriate one. In address recognition, the classifier receives the features extracted from the sub-image containing just one character and places it to one of the following classes: 'A','B','C'..., '0','1',...,'9'. The classifier can be thought as a mapping from the feature space to the set of possible classes. Note that the classifier cannot distinguish between two objects with the same feature vector.

The main content of this course is within the classifier design. There is a simple reason: The abstraction offered by the concept of the feature vector makes it possible to develop a general, application-independent theory for the design of classifiers. Therefore, it is possible to use the same underlying principles when designing classifiers for address recognition systems and bottle recycling machines.

Post-processing. A pattern recognition system rarely exists in a vacuum. The final task of the pattern recognition system is to decide upon an action based on the classification result(s). A simple example is a bottle recycling machine, which places bottles and cans to correct boxes for further processing. Different actions can have also different costs associated with them. This information can be included to the classifier design as we will see later on. Also, we can have several interdependent classification results in our hands. For example in an address recognition task, we have the classification results for multiple characters and the task is to decide upon the address that these characters form. Therefore, it is possible to use the *context*: In this case, the information about the other classification results to correct a possible misclassification. If the result of the classifier is 'Hollywoud Boulevard' then the address is probably 'Hollywood Boulevard'.

2.3 Design of Pattern Recognition Systems

The design of a pattern recognition system is an iterative process. If the system is not good enough, we go back to the drawing table and try to improve some or all of the five stages of the system (see the previous section 2.2). After that the system is tested again and it is improved if it still does not meet the requirements placed for it. The process is repeated until the system meets the requirements. The design process is founded on the knowledge about the particular pattern recognition application, the theory of pattern recognition and simply trial and error. Clearly, the smaller the importance of 'trial and error' the faster is the design cycle. This makes it important to minimize the amount of trial and error in the design of the pattern recognition system. The importance of 'trial and error' can be minimized by having good knowledge about the theory of pattern recognition and integrating that with knowledge about the application area. When is a pattern recognition system good enough? This obviously depends on the requirements of the application, but there are some general guidelines about how to evaluate the system. We will return to these later on during this course. How can a system be improved? For example, it can be possible to add a new feature to feature vectors to improve the classification accuracy. However, this is not always possible. For example with the Fisher's Iris dataset, we are restricted to the four existing features because we cannot influence the original measurements. Another possibility to improve a pattern recognition system is to improve the classifier. For this too there exists a variety of different means and it might not be a simple task to pick the best one out of these. We will return to this dilemma once we have the required background.

When designing a pattern recognition system, there are other factors that need to be considered besides the classification accuracy. For example, the time spent for classifying an object might be of crucial importance. The cost of the system can be an important factor as well.

2.4 Supervised and Unsupervised Learning and Classification

The classifier component of a pattern recognition system has to be taught to identify certain feature vectors to belong to a certain class. The points here are that 1) it is impossible to define the correct classes for all the possible feature vectors[1] and 2)

[1]Think about 16 times 16 binary images where each pixel is a feature, which has either value 0 or 1. In this case, we have $2^{256} \approx 10^{77}$ possible feature vectors. To put it more plainly, we have more feature vectors in the feature space than there are atoms in this galaxy.

the purpose of the system is to assign an object *which it has not seen previously* to a correct class.

It is important to distinguish between two types of machine learning when considering the pattern recognition systems. The (main) learning types are *supervised learning* and *unsupervised learning*. We also use terms *supervised classification* and *unsupervised classification*

In supervised classification, we present examples of the correct classification (a feature vector along with its correct class) to teach a classifier. Based on these examples, that are sometimes termed *prototypes* or *training samples*, the classifier then learns how to assign an unseen feature vector to a correct class. The generation of the prototypes (i.e. the classification of feature vectors/objects they represent) has to be done manually in most cases. This can mean lot of work: After all, it was because we wanted to avoid the hand-labeling of objects that we decided to design a pattern recognition system in the first place. That is why the number of prototypes is usually very small compared to the number of possible inputs received by the pattern recognition system. Based on these examples we would have to deduct the class of a never seen object. Therefore, the classifier design must be based on the assumptions made about the classification problem in addition to prototypes used to teach the classifier. These assumptions can often be described best in the language of probability theory.

In unsupervised classification or *clustering*, there is no explicit teacher nor training samples. The classification of the feature vectors must be based on similarity between them based on which they are divided into natural groupings. Whether any two feature vectors are similar depends on the application. Obviously, unsupervised classification is a more difficult problem than supervised classification and supervised classification is the preferable option if it is possible. In some cases, however, it is necessary to resort to unsupervised learning. For example, this is the case if the feature vector describing an object can be expected to change with time.

There exists a third type of learning: *reinforcement learning*. In this learning type, the teacher does not provide the correct classes for feature vectors but the teacher provides feedback whether the classification was correct or not. In OCR, assume that the correct class for a feature vector is 'R'. The classifier places it into the class 'B'. In reinforcement learning, the feedback would be that 'the classification is incorrect'. However the feedback does not include any information about the correct class.

Chapter 3

Background on Probability and Statistics

Understanding the basis of the probability theory is necessary for the understanding of the statistical pattern recognition. After all we aim to derive a classifier that makes as few errors as possible. The concept of the error is defined statistically, because there is variation among the feature vectors representing a certain class. Very far reaching results of the probability theory are not needed in an introductory pattern recognition course. However, the very basic concepts, results and assumptions of the probability theory are immensely useful. The goal of this chapter is to review the basic concepts of the probability theory.

3.1 Examples

The probability theory was born due to the needs of gambling. Therefore, it is pertinent to start with elementary examples related to gambling.

What is the probability that a card drawn from a well-shuffled deck of cards is an ace?

In general, the probability of an *event* is the number of the favorable outcomes divided by the total number of all possible outcomes. In this case, there are 4 favorable outcomes (aces). The total number of outcomes is the number of cards in the deck, that is 52. The probability of the event is therefore $4/52 = 1/13$.

Consider next rolling two dice at the same time. The total number of outcomes is 36. What is then the probability that the sum of the two dice is equal to 5? The number of the favorable outcomes is 4 (what are these?). And the probability is 1/9.

3.2 Basic Definitions and Terminology

The set S is a *sample space* for an experiment (for us a measurement), if every physical outcome of the experiment refers to a unique element of S. These elements of S are termed *elementary events*.

A subset of S is called *an event*. An event is a set of possible outcomes of an experiment. A single performance of an experiment is a *trial*. At each trial we observe a single elementary event. We say that an event A *occurs* in the trial if the outcome of the trial belongs to A. If two events do not intersect, i.e. their intersection is empty, they are called *mutually exclusive*. More generally, the events $E_1, E_2, \ldots, E_n$ are mutually exclusive if $E_i \cap E_j = \emptyset$ for any $i \neq j$. Given two mutually exclusive events, A and B, the occurrence of A in a trial prevents B from occurring in the same trial and vice versa.

A probability measure assigns a probability to each event of the sample space. It expresses how frequently an outcome (an elementary event) belonging to an event occurs within the trials of an experiment. More precisely, a *probability measure* P on S is a real-valued function defined on the events of S such that:

1. For any event $E \subseteq S$, $0 \leq P(E) \leq 1$.
2. $P(S) = 1$.
3. If events $E_1, E_2, \ldots, E_i, \ldots$ are mutually exclusive

$$P(\bigcup_{i=1}^{\infty} E_i) = P(E_1) + P(E_2) + \cdots$$

These s.c. Kolmogorov axioms have natural interpretations (frequency interpretations). The axiom 1 gives the bounds for the frequency of occurrence as a real number. For example, a negative frequency of occurrence would not be reasonable. The axiom 2 states that in every trial something occurs. The axiom 3 states that if the events are mutually exclusive, the probability that one of the events occurs in a trial is the sum of the probabilities of events. Note that in order to this make any sense it is necessary that the events are mutually exclusive. [1]

Example. In the example involving a deck of cards, a natural sample space is the set of all cards. A card is a result of a trial. The event of our interest was the set of aces. The probability of each elementary event is 1/52 and the probabilities of other events can be computed based on this and the Kolmogorov axioms.

3.3 Properties of Probability Spaces

Probability spaces and measures have some useful properties, which can be proved from the three axioms only. To present (and prove) these properties one must

[1]More precisely the probability space is defined as a three-tuple $(S, \mathcal{F}, P)$, where $\mathcal{F}$ a set of subsets of S satisfying certain assumptions (s.c. Borel-field) where the mapping P is defined. The definitions as precise as this are not needed in this course. The axiomatic definition of the probability was conceived because there are certain problems with the classical definition utilized in the examples of the previous section. (e.g. Bertnand's paradox).

rely mathematical abstraction of the probability spaces offered by their definition. However, as we are more interested in their interpretation, a little dictionary is needed: In the following E and F are events and E^c denotes the complement of E.

set theory	probability
E^c	E does not happen
$E \cup F$	E or F happens
$E \cap F$	E and F both happen

Note that often $P(E \cap F)$ is abbreviated as $P(E, F)$.

And some properties:

Theorem 1 *Let E and F be events of the sample space S. The probability measure P of S has the following properties:*

1. $P(E^c) = 1 - P(E)$
2. $P(\emptyset) = 0$
3. *If E is a sub-event of F:n ($E \subseteq F$), then* $P(F - E) = P(F) - P(E)$
4. $P(E \cup F) = P(E) + P(F) - P(E \cap F)$

If the events $F_1, \dots, F_n$ are mutually exclusive and $S = F_1 \cup \dots \cup F_n$, then

5. $P(E) = \sum_{i=1}^{n} P(E \cap F_i)$.

A collection of events as in the point 5 is called *a partition of S*. Note that an event F and its complement F^c always form a partition. The proofs of the above theorem are simple and they are skipped in these notes. The *Venn diagrams* can be useful if the logic behind the theorem seems fuzzy.

3.4 Random Variables and Random Vectors

Random variables characterize the concept of random experiment in probability theory. In pattern recognition , the aim is to assign an object to a correct class based on the observations about the object. The observations about an object from a certain class can be thought as a trial. The idea behind introducing the concept of a random variable is that, via it, we can transfer all the considerations to the space of real numbers $\mathbb{R}$ and we do not have to deal separately with separate probability spaces. In practise, everything we need to know about random variable is its probability distribution (to be defined in next section).

Formally, a random variable (RV) is defined as a real-valued function X defined on a sample space S, i.e. $X : S \to \mathbb{R}$. The set of values $\{X(x) : x \in S\}$ taken on by X is called the *co-domain* or the range of X.

Example. Recall our first example, where we wanted to compute the probability of drawing an ace from a deck of cards. The sample space consisted of the labels of the cards. A relevant random variable in this case is a map X : labels of cards $\to \{0, 1\}$, where $X(\text{ace}) = 1$ and $X(\text{notace}) = 0$, where notace refers to all the cards that are not aces. When rolling two dice, a suitable RV is e.g. a map from the sample space to the sum of two dice.

These are examples of relevant RVs only. There is no unique RV that can be related to a certain experiment. Instead, there exist several possibilities, some of which are better than others.

Random vectors are generalizations of random variables to multivariate situations. Formally, these are defined as maps $X : S \to \mathbb{R}^d$, where d is an integer. For random vectors, the definition of co-domain is a natural extension of the definition of co-domain for random variables. In pattern recognition, the classification seldom succeeds if we have just one observation/measurement/feature in our disposal. Therefore, we are mainly concerned with random vectors. From now on, we do not distinguish between a single variate random variable and a random vector. Both are called random variables unless there is a special need to emphasize multidimensionality.

We must still connect the concept of the random variable to the concept of the probability space. The task of random variables is to transfer the probabilities defined for the subsets of the sample space to the subsets of the Euclidean space: The probability of a subset A of the co-domain of X is equal to the probability of inverse image of A in the sample space. This makes it possible to model probabilities with the probability distributions.

Often, we can just 'forget' about the sample space and model the probabilities via random variables. The argument of the random variable is often dropped from the notation, i.e. $P(X \in B)$ is equivalent to $P(\{s : X(s) \in B, s \in S, B \subseteq \mathbb{R}^d\})$[2].

Note that a value obtained by a random variable reflects a result of a single trial. This is different than the random variable itself. These two should under no circumstances be confused although the notation may give possibility to do that.

3.5 Probability Densities and Cumulative Distributions

3.5.1 Cumulative Distribution Function (cdf)

First we need some additional notation: For vectors $\mathbf{x} = [x_1, \ldots, x_d]^T$ and $\mathbf{y} = [y_1, \ldots, y_d]^T$ notation $\mathbf{x} < \mathbf{y}$ is equivalent to $x_1 < y_1$ and $x_2 < y_2$ and $\cdots$ and $x_d < y_d$. That is, if the components of the vectors are compared pairwise then the components of $\mathbf{x}$ are all smaller than components of $\mathbf{y}$.

[2]For a good and more formal treatment of the issue, see E.R. Dougherty: Probability and Statistics for the Engineering, Computing and Physical Sciences.

The cumulative distribution function F_X (cdf) of an RV X is defined for all $\mathbf{x}$ as

$$F_X(\mathbf{x}) = P(X \leq \mathbf{x}) = P(\{s : X(s) \leq \mathbf{x}\}). \tag{3.1}$$

The cdf measures the probability mass of all $\mathbf{y}$ that are smaller than $\mathbf{x}$.

Example. Let us once again consider the deck of cards and the probability of drawing an ace. The RV in this case was the map from the labels of the cards to the set $\{0, 1\}$ with the value of the RV equal to 1 in the case of a ace and 0 otherwise. Then $F_X(x) = 0$, when $x < 0$, $F_X(x) = 48/52$, when $0 \leq x < 1$, and $F_X(x) = 1$ otherwise.

Theorem 2 *The cdf F_X has the following properties:*

1. *F_X is increasing, i.e. if $\mathbf{x} \leq \mathbf{y}$, then $F_X(\mathbf{x}) \leq F_X(\mathbf{y})$.*
2. *$\lim_{\mathbf{x} \to -\infty} F_X(\mathbf{x}) = 0$, $\lim_{\mathbf{x} \to \infty} F_X(x) = 1$.*

3.5.2 Probability Densities: Discrete Case

An RV X is *discrete* if its co-domain is denumerable. (The elements denumerable set can be written as a list $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$. This list may be finite or infinite).

The probability density function (pdf) p_X of a discrete random variable X is defined as

$$p_X(\mathbf{x}) = \begin{cases} P(X = \mathbf{x}) & \text{if } \mathbf{x} \text{ belongs to codomain of } X \\ 0 & \text{otherwise} \end{cases}. \tag{3.2}$$

In other words, a value of the pdf $p_X(\mathbf{x})$ is the probability of the event $X = \mathbf{x}$. If an RV X has a pdf p_X, we say that X is distributed according to p_X and we denote $X \sim p_X$. The cdf of a discrete RV can we then written as

$$F_X(\mathbf{x}) = \sum_{\mathbf{y} \leq \mathbf{x}} p_X(\mathbf{y}),$$

where the summation is over $\mathbf{y}$ that belong to the co-domain of X.

3.5.3 Probability Densities: Continuous Case

An RV X is *continuous* if there exists such function p_X that

$$F_X(\mathbf{x}) = \int_{-\infty}^{\mathbf{x}} p_X(\mathbf{y}) d\mathbf{y}. \tag{3.3}$$

the function p_X is then *the probability density function* of X. In Figure 3.1, the pdf and the cdf of the standard normal distribution are depicted.

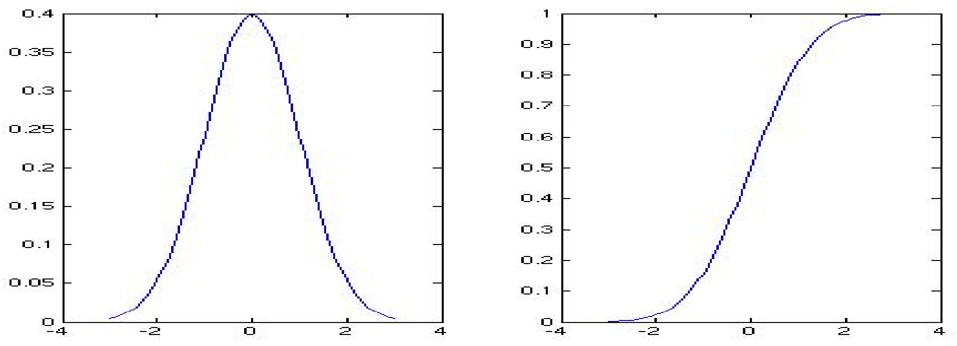

Figure 3.1: The pdf (left) and the cdf (right) of the standard normal distribution.

The probability that the occurrence $\mathbf{x}$ of an RV X belongs to a subset B of $\mathbb{R}^d$ is

$$P(X \in B) = \int_B p_X(\mathbf{y}) d\mathbf{y}.$$

In the continuous case, the value of pdf $p_X(\mathbf{x})$ is not a probability: The value of any integral over a single point is zero and therefore it is not meaningful to study the value of the probability of occurrence of a single value of an RV in the continuous case.

Some notes about integrals.

- During this course, it is assumed that the integrals are multivariate Riemann integrals.
- The integral
$$\int_{-\infty}^{\mathbf{x}} p_X(\mathbf{y}) d\mathbf{y}$$
is an abbreviation for
$$\int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_d} p_X(y_1, y_2, \ldots, y_d) dy_1 \cdots dy_d,$$
where $\mathbf{x} = [x_1, \ldots, x_d]^T$ and $\mathbf{y} = [y_1, \ldots, y_d]^T$
- And most importantly: you do not have to evaluate any integrals in practical pattern recognition. However, the basic properties of integrals are useful to know.

The following theorem characterizes the pdfs:

Theorem 3 *A function $p : \mathbb{R}^d \to \mathbb{R}$ is the probability density for a continuous RV if and only if*

1. $p(\mathbf{x}) \geq 0$ *for all* $\mathbf{x}$.
2. $\int_{-\infty}^{\infty} p(\mathbf{x}) d\mathbf{x} = 1$.

Let D be the co-domain of a discrete RV X. A function $p : D \to \mathbb{R}$ is the probability density for a discrete RV X if and only if

1. *$p(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ within the co-domain of X.*

2. *$\sum_{\mathbf{x}} p(\mathbf{x}) = 1$, where the summation runs over the co-domain of X.*

The above theorem is a direct consequence of the Kolmogorov axioms. Also, the pdf (if it exists) defines uniquely the random variable. This means that in most cases all we need is the knowledge about the pdf of an RV.

Finally, the probability density function is the derivative of the cumulative distribution function if the cdf is differentiable. In the multivariate case, the pdf is obtained based on the cdf by differentiating the cdf with respect to all the components of the vector valued variate.

3.6 Conditional Distributions and Independence

3.6.1 Events

Assume that the events E and F relate to the same experiment. The events $E \subset S$ and $F \subset S$ are *(statistically) independent* if

$$P(E \cap F) = P(E)P(F). \tag{3.4}$$

If E and F are not independent, they are *dependent*. The independence of events E and F means that an occurrence of E does not affect the likelihood of an occurrence of F in the same experiment. Assume that $P(F) \neq 0$. The *conditional probability of E relative to $F \subseteq S$* is denoted by $P(E|F)$ and defined as

$$P(E|F) = \frac{P(E \cap F)}{P(F)}. \tag{3.5}$$

This refers to the probability of the event E provided that the event F has already occurred. In other words, we know that the result of a trial was included in the event F.

If E and F are independent, then

$$P(E|F) = \frac{P(E \cap F)}{P(F)} = \frac{P(E)P(F)}{P(F)} = P(E).$$

Accordingly, if $P(E|F) = P(E)$, then

$$P(E \cap F) = P(E)P(F), \tag{3.6}$$

and events E and F are independent. The point is that an alternative definition of independence can be given via conditional probabilities. Note also that the concept of independence is different from the concept of mutual exclusivity. Indeed, if the events E and F are mutually exclusive, then $P(E|F) = P(F|E) = 0$, and E and F are dependent.

Definition (3.4) does not generalize in a straight-forward manner. The independence of the events $E_1, \ldots, E_n$ must be defined inductively. Let $K = \{i_1, i_2, \ldots, i_k\}$ be an arbitrary subset of the index set $\{1, \ldots, n\}$. The events $E_1, \ldots, E_n$ are independent if it holds for every $K \subset \{1, \ldots, n\}$ that

$$P(E_{i_1} \cap \cdots \cap E_{i_k}) = P(E_{i_1}) \cdots P(E_{i_k}). \tag{3.7}$$

Assume that we have more than two events, say E, F and G. They are pairwise independent if E and F are independent and F and G are independent and E and G are independent. However, it does not follow from the pairwise independence that E, F and G would be independent. The definition of the statistical independence is more easily defined by random variables and their pdfs as we shall see in the next subsection.

3.6.2 Random Variables

Let us now study the random variables $X_1, X_2, \ldots, X_n$ related to the same experiment. We define a random variable

$$X = [X_1^T, X_2^T, \ldots, X_n^T]^T.$$

(X is a vector of vectors). The pdf

$$p_X(\mathbf{x}) = p_{(X_1,\ldots,X_n)}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$$

is the *joint probability density function* of the RVs $X_1, X_2, \ldots, X_n$. The value

$$p_{(X_1,\ldots,X_n)}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$$

refers to the probability density of X_1 receiving the value $\mathbf{x}_1$, X_2 receiving the value $\mathbf{x}_2$ and so forth in a trial. The pdfs $p_{X_i}(\mathbf{x}_i)$ are called *the marginal densities* of X. These are obtained from the joint density by integration. The joint and marginal distributions have all the properties of pdfs.

The random variables $X_1, X_2, \ldots, X_n$ are *independent* if and only if

$$p_X(\mathbf{x}) = p_{(X_1,\ldots,X_n)}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = p_{X_1}(\mathbf{x}_1) \cdots p_{X_n}(\mathbf{x}_n). \tag{3.8}$$

Let E_i $i = 1, \ldots, n$ and $F_j, j = 1, \ldots, m$ be events related to X_1 and X_2, respectively. Assume that X_1 and X_2 are independent, Then also all the events related to the RVs X_1 and X_2 are independent, i.e.

$$P(E_i \cap F_j) = P(E_i)P(F_j),$$

for all i and j. That is to say that the results of the (sub)experiments modeled by independent random variables are not dependent on each other. The independence of the two (or more) RVs can be defined with the help of the events related to the RVs. This definition is equivalent to the definition by the pdfs.

It is natural to attach the concept of conditional probability to random variables. We have RVs X and Y. A new RV modeling the probability of X assuming that we already know that $Y = \mathbf{y}$ is denoted by $X|\mathbf{y}$. It is called the *conditional random variable of* X *given* $\mathbf{y}$. The RV $X|\mathbf{y}$ has a density function defined by

$$p_{X|Y}(\mathbf{x}|\mathbf{y}) = \frac{p_{X,Y}(\mathbf{x}, \mathbf{y})}{p_Y(\mathbf{y})}. \tag{3.9}$$

RVs X and Y are independent if and only if

$$p_{X|Y}(\mathbf{x}, \mathbf{y}) = p_X(\mathbf{x})$$

for all $\mathbf{x}, \mathbf{y}$.

3.7 Bayes Rule

This section is devoted to a very important statistical result to pattern recognition. This *Bayes Rule* was named after Thomas Bayes (1702 - 1761). This rule has versions concerning events and RVs, which both are represented. The derivations here are carried out only for the case concerning events.

We shall now derive the Bayes rule. Multiplying by $P(F)$ the both sides of the definition of the conditional probability for events E and F yields

$$P(E \cap F) = P(F)P(E|F).$$

For this we have assumed that $P(F) > 0$. If also $P(E) > 0$, then

$$P(E \cap F) = P(E)P(F|E).$$

Combining the two equations yields

$$P(F)P(E|F) = P(E \cap F) = P(E)P(F|E).$$

And furthermore

$$P(F|E) = \frac{P(F)P(E|F)}{P(E)}.$$

This is the Bayes rule. A different formulation of the rule is obtained via the point 5 in the Theorem 1. If the events $F_1, \ldots, F_n$ are mutually exclusive and the sample space $S = F_1 \cup \cdots \cup F_n$, then for all k it holds that

$$P(F_k|E) = \frac{P(F_k)P(E|F_k)}{P(E)} = \frac{P(F_k)P(E|F_k)}{\sum_{i=1}^{n} P(F_i)P(E|F_i)}.$$

We will state these results and the corresponding ones for the RVs in a theorem:

Theorem 4 *Assume that $E, F, F_1, \ldots, F_n$ are events of S. Moreover $F_1, \ldots, F_n$ form a partition of S, and $P(E) > 0$, $P(F) > 0$. Then*

1. $P(F|E) = \frac{P(F)P(E|F)}{P(E)}$;

2. $P(F_k|E) = \frac{P(F_k)P(E|F_k)}{\sum_{i=1}^{n} P(F_i)P(E|F_i)}$.

Assume that X and Y are RVs related to the same experiment. Then

3. $p_{X|Y}(\mathbf{x}|\mathbf{y}) = \frac{p_X(\mathbf{x})p_{Y|X}(\mathbf{y}|\mathbf{x})}{p_Y(\mathbf{y})}$;

4. $p_{X|Y}(\mathbf{x}|\mathbf{y}) = \frac{p_X(\mathbf{x})p_{Y|X}(\mathbf{y}|\mathbf{x})}{\int p_X(\mathbf{x})p_{Y|X}(\mathbf{y}|\mathbf{x})d\mathbf{x}}$.

Note that the points 3 and 4 hold even if one of the RVs was discrete and the other was continuous. Obviously, it can be necessary to replace the integral by a sum.

3.8 Expected Value and Variance

We take a moment to define the concepts of *expected value* and *variance*, because we will later on refer to these concepts. The expected value of the continuous RV X is defined as

$$E[X] = \int_{-\infty}^{\infty} \mathbf{x} p_X(\mathbf{x}) d\mathbf{x}.$$

The expected value of the discrete RV X is defined as

$$E[X] = \sum_{\mathbf{x} \in D_X} \mathbf{x} p_X(\mathbf{x}),$$

where D_X is co-domain of X. Note that if X is a d-component RV then also $E[X]$ is a d-component vector.

The variance of the continuous RV X is defined as

$$Var[X] = \int_{-\infty}^{\infty} (\mathbf{x} - E[X])(\mathbf{x} - E[X])^T p_X(\mathbf{x}) d\mathbf{x}.$$

The variance of the discrete RV X is defined as

$$Var[X] = \sum_{\mathbf{x} \in D_X} (\mathbf{x} - E[X])(\mathbf{x} - E[X])^T p_X(\mathbf{x}),$$

where D_X is co-domain of X. The variance of X is a $d \times d$ matrix. In particular, if $X \sim N(\mu, \Sigma)$, then $E[X] = \mu$ and $Var[X] = \Sigma$.

3.9 Multivariate Normal Distribution

The multivariate normal distribution is widely used in pattern recognition and statistics. Therefore, we have a closer look into this distribution. First, we need the definition for positive definite matrices: Symmetric $d \times d$ matrix A is positive-definite if for all non-zero $\mathbf{x} \in \mathbb{R}^d$ it holds that

$$\mathbf{x}^T A\mathbf{x} > 0.$$

Note that, for the above inequality to make sense, the value of *the quadratic form* $\mathbf{x}^T A\mathbf{x} = \sum_{i=1}^{d}\sum_{j=1}^{d} a_{ij}x_i x_j$ must be a scalar. This implies that $\mathbf{x}$ must be a column-vector (i.e. a tall vector). A positive definite matrix is always non-singular and thus it has the inverse matrix. Additionally, the determinant of a positive definite matrix is always positive.

Example. The matrix

$$B = \begin{bmatrix} 2 & 0.2 & 0.1 \\ 0.2 & 3 & 0.5 \\ 0.1 & 0.5 & 4 \end{bmatrix}$$

is positive-definite. We illustrate the calculation of the value of a quadratic form by selecting $\mathbf{x} = \begin{bmatrix} -2 & 1 & 3 \end{bmatrix}^T$. Then

$$\begin{aligned} \mathbf{x}^T B\mathbf{x} &= 2\cdot(-2)\cdot(-2) + 0.2\cdot 1\cdot(-2) + 0.1\cdot 3\cdot(-2) \\ &+ 0.2\cdot(-2)\cdot 1 + 3\cdot 1\cdot 1 + 0.5\cdot 3\cdot 1 \\ &+ 0.1\cdot(-2)\cdot 3 + 0.5\cdot 1\cdot 3 + 4\cdot 3\cdot 3 \\ &= 48. \end{aligned}$$

A d-component RV $X \sim N(\mu, \Sigma)$ distributed according the normal distribution has a pdf

$$p_X(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\Sigma)}}\exp[-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)], \qquad (3.10)$$

where the parameter μ is a d component vector (the mean of X) and Σ is a $d \times d$ positive definite matrix (the covariance of X). Note that because Σ is positive definite, also Σ^{-1} is positive definite (the proof of this is left as an exercise). From this it follows that $\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)$ is always non-negative and zero only when $\mathbf{x} = \mu$. This is important since it ensures that the pdf is bounded from above. The mode of the pdf is located at μ. The determinant $\det(\Sigma) > 0$ and therefore $p_X(\mathbf{x}) > 0$ for all $\mathbf{x}$. Also, $\int p_X(\mathbf{x})d\mathbf{x} = 1$ (the proof of this is skipped, for the 1-dimensional case, see e.g. `http://mathworld.wolfram.com/GaussianIntegral.html`). Hence, the density in (3.10) is indeed a proper probability density.

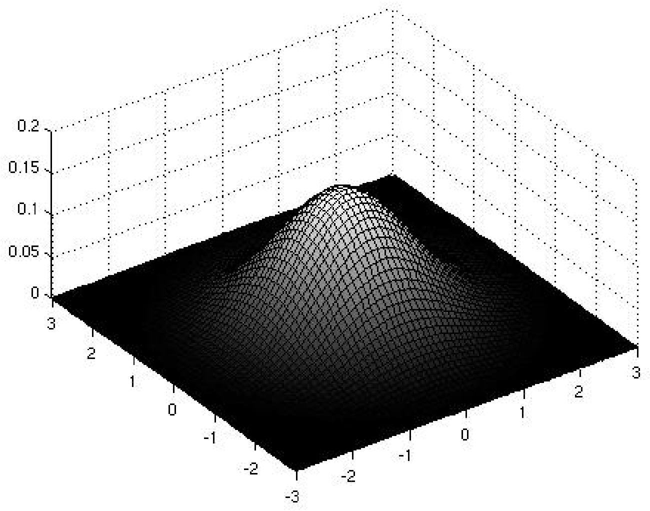

Figure 3.2: The pdf of a 2-dimensional normal distribution

In the univariate case, the pdf is

$$p_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp[-\frac{1}{2}(\frac{x-\mu}{\sigma})^2].$$

Example. Let us calculate the value of a normal pdf at $\mathbf{x}$ to continue the above example concerning quadratic forms. Assume that $\mathbf{x} = [-1, 2, 4]^T, \mu = [1, 1, 1]^T$ and

$$\Sigma = \frac{1}{23.33} \begin{bmatrix} 11.75 & -0.75 & -0.20 \\ -0.75 & 7.99 & -0.98 \\ -0.20 & -0.98 & 5.96 \end{bmatrix}.$$

Hence (here we need Matlab),

$$\Sigma^{-1} = \begin{bmatrix} 2 & 0.2 & 0.1 \\ 0.2 & 3 & 0.5 \\ 0.1 & 0.5 & 4 \end{bmatrix}$$

Now, $\mathbf{x} - \mu = [-2, 1, 3]^T$, and based on the previous example

$$s = (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) = 48$$

and

$$\exp(-0.5s) \approx 3.77 \cdot 10^{-11}.$$

The determinant $\det(\Sigma) = 1/23.33$ (again we used Matlab). The value of the pdf

$$p_X([-1, 2, 4]^T) \approx (\sqrt{23.33}/15.7496) \cdot 3.77 \cdot 10^{-11} = 1.16 \cdot 10^{-11},$$

which is extremely small value. Two notes are in order: 1) Remember that the value of a pdf at a single point is not a probability of any event. However, computing values of pdfs like this one is commmon in pattern recognition as we will see in the next chapter. 2) All calculations like these are performed numerically and using a computer, the manual computations presented here are just for illustration.

The normal distribution has some interesting properties that make it special among the distributions of continuous random variables. Some of them are worth of knowing for this course. In what follows, we denote the components of μ by μ_i and the components of Σ by σ_{ij}. If $X \sim N(\mu, \Sigma)$ then

- sub-variables $X_1, \ldots, X_d$ of X are normally distributed: $X_i \sim N(\mu_i, \sigma_{ii})$;
- sub-variables X_i and X_j of X are independent if and only if $\sigma_{ij} = 0$.
- Let A and B be non-singular $d \times d$ matrices. Then $AX \sim N(A\mu, A\Sigma A^T)$ and RVs AX and BX are independent if and only if $A\Sigma B^T$ is the zero-matrix.
- The sum of two normally distributed RVs is normally distributed.

Chapter 4

Bayesian Decision Theory and Optimal Classifiers

The Bayesian decision theory offers a solid foundation for classifier design. It tells us how to design the optimal classifier when the statistical properties of the classification problem are known. The theory is a formalization of some common-sense principles, but it offers a good basis for classifier design.

About notation. In what follows we are a bit more sloppy with the notation than previously. For example, pdfs are not anymore indexed with the symbol of the random variable they correspond to.

4.1 Classification Problem

Feature vectors $\mathbf{x}$ that we aim to classify belong to the *feature space.* The feature space is usually d-dimensional Euclidean space $\mathbb{R}^d$. However, the feature space can be for instance $\{0, 1\}^d$, the space formed by the d-bit binary numbers. We denote the feature space with the symbol $\mathbb{F}$.

The task is to assign an arbitrary feature vector $\mathbf{x} \in \mathbb{F}$ to one of the c classes[1] The c classes are denoted by $\omega_1, \ldots, \omega_c$ and these are assumed to be known. Note that we are studying a pair of random variables (X, ω), where X models the value of the feature vector and ω models its class.

We know the

1. *prior probabilities* $P(\omega_1), \ldots, P(\omega_c)$ of the classes and
2. the *class conditional probability density functions* $p(\mathbf{x}|\omega_1), \ldots, p(\mathbf{x}|\omega_c)$.

[1]To be precise, the task is to assign the object that is represented by the feature vector to a class. Hence, what we have just said is not formally true. However, it is impossible for the classifier to distinguish between two objects with the same feature vector and it is reasonable to speak about the feature vectors instead of the object they represent. This is also typical in the literature. In some cases, this simplification of the terminology might lead to misconceptions and in these situations we will use more formal terminology.

The prior probability $P(\omega_i)$ defines what percentage of all feature vectors belong to the class ω_i. The class conditional pdf $p(\mathbf{x}|\omega_i)$, as it is clear from the notation, defines the pdf of the feature vectors belonging to ω_i. This is same as the density of X given that the class is ω_i. To be precise: The marginal distribution of ω is known and the distributions of X given ω_i are known for all i. The general laws of probability obviously hold. For example,

$$\sum_{i=1}^{c} P(\omega_i) = 1.$$

We denote a classifier by the symbol α. Because the classification problem is to assign an arbitrary feature vector $\mathbf{x} \in \mathbb{F}$ to one of c classes, the classifier is a function from the feature space onto the set of classes, i.e. $\alpha : \mathbb{F} \rightarrow \{\omega_1, \ldots, \omega_c\}$. The classification result for the feature vector $\mathbf{x}$ is $\alpha(\mathbf{x})$. We often call a function α a *decision rule.*

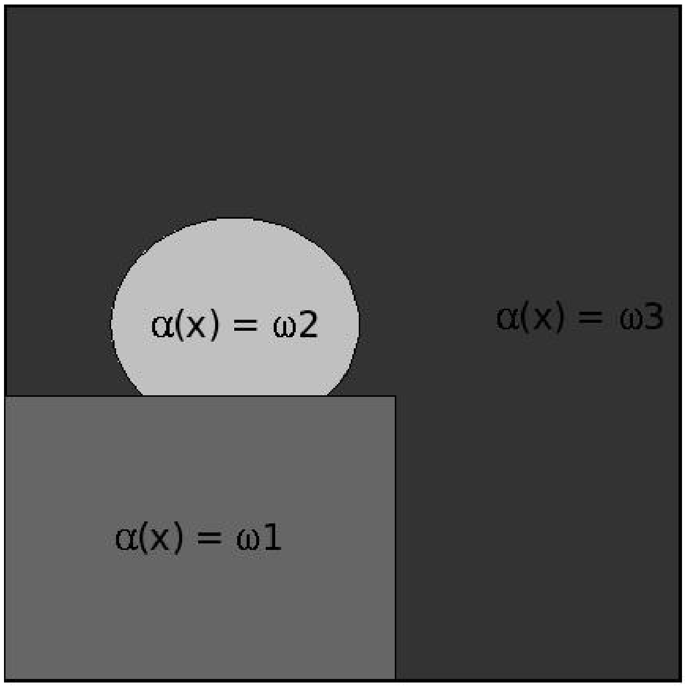

Figure 4.1: Decision regions

The consideration of the partition of the feature space generated by a classifier leads to another interpretation for the classifier. Every feature vector is assigned to a unique class by a classifier and therefore the classifier defines those feature vectors that belong to a certain class, see Figure 4.1. The parts of the feature space corresponding to classes $\omega_1, \ldots, \omega_c$ are denoted by $\mathcal{R}_1, \ldots, \mathcal{R}_c$, respectively. The sets $\mathcal{R}_i$ are called *decision regions.* Clearly,

$$\mathcal{R}_i = \{\mathbf{x} : \alpha(\mathbf{x}) = \omega_i\}.$$

The decision regions form a partition of the feature space (i.e. $\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ when $i \neq j$ and $\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_c = \mathbb{F}$) As the decision rule defines the decision regions, the decision regions define the decision rule: The classifier can be represented by its decision regions. However, especially for large d, the representation by decision regions is not convenient. However, the representation by decision regions often offers additional intuition on how the classifier works.

4.2 Classification Error

The classification problem is by nature statistical. Objects having same feature vectors can belong to different classes. Therefore, we cannot assume that it would be possible to derive a classifier that works perfectly. (The perfect classifier would assign every object to the correct class). This is why we must study the classification error: For a specified classification problem our task to derive classifier that makes as few errors (misclassifications) as possible. We denote the classification error by the decision rule α by $E(\alpha)$. Because the classifier α assigns an object to a class only based on the feature vector of that object, we study the probability $E(\alpha(\mathbf{x})|\mathbf{x})$ that objects with the feature vector $\mathbf{x}$ are misclassified.

Let us take a closer look into the classification error and show that it can be computed if the preliminary assumptions of the previous section hold. From the item 1 in Theorem 1, it follows that

$$E(\alpha(\mathbf{x})|\mathbf{x}) = 1 - P(\alpha(\mathbf{x})|\mathbf{x}), \tag{4.1}$$

where $P(\alpha(\mathbf{x})|\mathbf{x})$ is the probability that the class $\alpha(\mathbf{x})$ is correct for the object. The classification error for the classifier α can be written as

$$E(\alpha) = \int_{\mathbb{F}} E(\alpha(\mathbf{x})|\mathbf{x})p(\mathbf{x})d\mathbf{x} = \int_{\mathbb{F}} [1 - P(\alpha(\mathbf{x})|\mathbf{x})]p(\mathbf{x})d\mathbf{x}. \tag{4.2}$$

We need to still know $P(\alpha(\mathbf{x})|\mathbf{x})$. From the Bayes rule we get

$$P(\alpha(\mathbf{x})|\mathbf{x}) = \frac{p(\mathbf{x}|\alpha(\mathbf{x}))P(\alpha(\mathbf{x}))}{p(\mathbf{x})}. \tag{4.3}$$

And hence the classification error is

$$E(\alpha) = 1 - \int_{\mathbb{F}} p(\mathbf{x}|\alpha(\mathbf{x}))P(\alpha(\mathbf{x}))d\mathbf{x}, \tag{4.4}$$

where $p(\mathbf{x}|\alpha(\mathbf{x})), P(\alpha(\mathbf{x}))$ are known for every $\mathbf{x}$ as it was defined in the previous section. Note that

$$p(\mathbf{x}|\alpha(\mathbf{x}))P(\alpha(\mathbf{x})) = p(\mathbf{x}, \alpha(\mathbf{x})).$$

That is, the classification error of α is equal to the probability of the complement of the event $\{(\mathbf{x}, \alpha(\mathbf{x})) : \mathbf{x} \in \mathbb{F}\}$.

With the knowledge of decision regions, we can rewrite the classification error as

$$E(\alpha) = \sum_{i=1}^{c} \int_{\mathcal{R}_i} [1 - p(\mathbf{x}|\omega_i)P(\omega_i)]d\mathbf{x} = 1 - \sum_{i=1}^{c} \int_{\mathcal{R}_i} p(\mathbf{x}|\omega_i)P(\omega_i)d\mathbf{x}, \tag{4.5}$$

where $\mathcal{R}_i, i = 1, \ldots, c$ are decision regions for the decision rule α.

4.3 Bayes Minimum Error Classifier

In this section, we study *Bayes minimum error classifier* which, provided that the assumptions of section 4.1 hold, minimizes the classification error. (The assumptions were that class conditional pdfs and prior probabilities are known.) This means that Bayes minimum error classifier, later on just Bayes classifier, is optimal for a given classification problem.

The Bayes classifier is defined as

$$\alpha_{Bayes}(\mathbf{x}) = \arg \max_{\omega_i, i=1,\ldots,c} P(\omega_i|\mathbf{x}). \tag{4.6}$$

Notation $\arg\max_{\mathbf{x}} f(\mathbf{x})$ means the value of the argument $\mathbf{x}$ that yields the maximum value for the function f. For example, if $f(x) = -(x-1)^2$, then $\arg\max_x f(x) = 1$ (and $\max f(x) = 0$). In other words,

> the Bayes classifier selects the most probable class when the observed feature vector is $\mathbf{x}$.

The posterior probability $P(\omega_i|\mathbf{x})$ is evaluated based on the Bayes rule, i.e. $P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})}$. Note additionally that $p(\mathbf{x})$ is equal for all classes and $p(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ and we can multiply the right hand side of (4.6) without changing the classifier. The Bayes classifier can be now rewritten as

$$\alpha_{Bayes}(\mathbf{x}) = \arg \max_{\omega_i, i=1,\ldots,c} p(\mathbf{x}|\omega_i)P(\omega_i). \tag{4.7}$$

If two or more classes have the same posterior probability given $\mathbf{x}$, we can freely choose between them. In practice, the Bayes classification of $\mathbf{x}$ is performed by computing $p(\mathbf{x}|\omega_i)P(\omega_i)$ for each class $\omega_i, i = 1,\ldots,c$ and assigning $\mathbf{x}$ to the class with the maximum $p(\mathbf{x}|\omega_i)P(\omega_i)$.

By its definition the Bayes classifier minimizes the conditional error $E(\alpha(\mathbf{x})|\mathbf{x}) = 1 - P(\alpha(\mathbf{x})|\mathbf{x})$ for all $\mathbf{x}$. Because of this and basic properties of integrals, the Bayes classifier minimizes also the classification error:

Theorem 5 *Let $\alpha : \mathbb{F} \to \{\omega_1,\ldots,\omega_c\}$ be an arbitrary decision rule and $\alpha_{Bayes} : \mathbb{F} \to \{\omega_1,\ldots,\omega_c\}$ the Bayes classifier. Then*

$$E(\alpha_{Bayes}) \leq E(\alpha).$$

The classification error $E(\alpha_{Bayes})$ of the Bayes classifier is called *the Bayes error*. It is the smallest possible classification error for a fixed classification problem. The Bayes error is

$$E(\alpha_{Bayes}) = 1 - \int_{\mathbb{F}} \max_i [p(\mathbf{x}|\omega_i)P(\omega_i)] d\mathbf{x} \tag{4.8}$$

Finally, note that the definition of the Bayes classifier does not require the assumption that the class conditional pdfs are Gaussian distributed. The class conditional pdfs can be any proper pdfs.

4.4 Bayes Minimum Risk Classifier

The Bayes minimum error classifier is a special case of the more general *Bayes minimum risk classifier.* In addition to the assumptions and terminology introduced previously, we are given a actions $\alpha_1, \ldots, \alpha_a$. These correspond to the actions that follow the classification. For example, a bottle recycling machine, after classifying the bottles and cans, places them in the correct containers and returns a receipt to the customer. The actions are tied to the classes via *a loss function* λ. The value $\lambda(\alpha_i|\omega_j)$ quantifies the loss incurred by taking an action α_i when the true class is ω_j. The decision rules are now functions from the feature space onto the set of actions. We need still one more definition. *The conditional risk* of taking the action α_i when the observed feature vector is $\mathbf{x}$ is defined as

$$R(\alpha_i|\mathbf{x}) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j) P(\omega_j|\mathbf{x}). \tag{4.9}$$

The definition of the Bayes minimum risk classifier is then:

> The Bayes minimum risk classifier chooses the action with the minimum conditional risk.

The Bayes minimum risk classifier is the optimal in this more general setting: it is the decision rule that minimizes the total risk given by

$$R_{total}(\alpha) = \int R(\alpha(\mathbf{x})|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}. \tag{4.10}$$

The Bayes classifier of the previous section is obtained when the action α_i is the classification to the class ω_i and the loss function is zero-one loss:

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

The number of actions a can be different from the number of classes c. This is useful e.g. when it is preferable that the pattern recognition system is able to separate those feature vectors that cannot be reliably classified.

Example. An example about spam filtering illustrates the differences between the minimum error and the minimum risk classification. An incoming e-mail is either a normal (potentially important) e-mail (ω_1) or a junk mail (ω_2). We have two actions: α_1 (keep the mail) and α_2 (put the mail to /dev/null). Because losing a normal e-mail is on average three times more painful than getting a junk mail into the inbox, we select a loss function

$$\begin{array}{ll} \lambda(\alpha_1|\omega_1) = 0 & \lambda(\alpha_1|\omega_2) = 1 \\ \lambda(\alpha_1|\omega_2) = 3 & \lambda(\alpha_2|\omega_2) = 0 \end{array}.$$

(You may disagree about the loss function.) In addition, we know that $P(\omega_1) = 0.4, P(\omega_2) = 0.6$. Now, an e-mail has been received and its feature vector is $\mathbf{x}$. Based on the feature vector, we have computed $p(\mathbf{x}|\omega_1) = 0.35, p(\mathbf{x}|\omega_2) = 0.65$.

The posterior probabilities are:

$$P(\omega_1|\mathbf{x}) = \frac{0.35 \cdot 0.4}{0.35 \cdot 0.4 + 0.65 \cdot 0.6} = 0.264;$$

$$P(\omega_2|\mathbf{x}) = 0.736.$$

For the minimum risk classifier, we must still compute the conditional losses:

$$R(\alpha_1|\mathbf{x}) = 0 \cdot 0.264 + 0.736 = 0.736,$$

$$R(\alpha_2|\mathbf{x}) = 0 \cdot 0.736 + 3 \cdot 0.264 = 0.792.$$

The Bayes (minimum error) classifier classifies the e-mail as spam but the minimum risk classifier still keeps it in the inbox because of the smaller loss of that action.

After this example, we will concentrate only on the minimum error classification. However, many of the practical classifiers that will be introduced generalize easily to the minimum risk case.

4.5 Discriminant Functions and Decision Surfaces

We have already noticed that the same classifier can be represented in several different ways, e.g. compare (4.6) and (4.7) which define the same Bayes classifier. As always, we would like this representation be as simple as possible. In general, a classifier can be defined with the help of a set of *discriminant functions*. Each class ω_i has its own discriminant function g_i that takes a feature vector $\mathbf{x}$ as an input. The classifier assigns the feature vector $\mathbf{x}$ to the class with the highest $g_i(\mathbf{x})$. In other words, the class is ω_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x})$$

for all $i \neq j$.

For the Bayes classifier, we can select discriminant functions as:

$$g_i(\mathbf{x}) = P(\omega_i|\mathbf{x}), i = 1, \ldots, c$$

or as

$$g_i(\mathbf{x}) = p(\mathbf{x}|\omega_i)P(\omega_i), i = 1, \ldots, c.$$

The word 'select' appears above for a reason: Discriminant functions for a certain classifier are not uniquely defined. Many different sets of discriminant functions may define exactly the same classifier. However, a set of discriminant functions define a unique classifier (but not uniquely). The next result is useful:

Theorem 6 *Let $f : \mathbb{R} \to \mathbb{R}$ be increasing, i.e. $f(x) < f(y)$ always when $x < y$. Then the following sets of discriminant functions*

$$g_i(\mathbf{x}), i = 1, \ldots, c$$

and

$$f(g_i(\mathbf{x})), i = 1, \ldots, c$$

define the same classifier.

This means that discriminant functions can be multiplied by positive constants, constants can be added to them or they can be replaced by their logarithms without altering the underlying classifier.

Later on, we will study discriminant functions of the type

$$g_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}.$$

These are called *linear discriminant functions.* Above $\mathbf{w}_i$ is a vector of the equal dimensionality with feature vector and w_{i0} is a scalar. A *linear classifier* is a classifier that can be represented by using only linear discriminant functions. Linear discriminant functions can be conceived as the most simple type of discriminant functions. This explains the interest in linear classifiers.

When the classification problem has just two classes, we can represent the classifier using a single discriminant function

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}). \tag{4.11}$$

If $g(\mathbf{x}) > 0$, then $\mathbf{x}$ is assigned to ω_1 and otherwise $\mathbf{x}$ is assigned to ω_2.

The classifier divides the feature space into parts corresponding classes. The decision regions can be defined with the help of discriminant functions as follows:

$$\mathcal{R}_i = \{\mathbf{x} |\ g_i(\mathbf{x}) > g_j(\mathbf{x}), i \neq j\}.$$

The boundaries between decision regions

$$\mathcal{R}_{ij} = \{\mathbf{x} : g_i(\mathbf{x}) = g_j(\mathbf{x}), g_k(\mathbf{x}) < g_i(\mathbf{x}), k \neq i, j\}$$

are called *decision surfaces.* It is easier to grasp the geometry of the classifier (i.e. the partition of the feature space by the classifier) based on decision surfaces than based on decision regions directly. For example, it easy to see that the decision surfaces of linear classifiers are segments of lines between the decision regions.

Example. Consider a two-class classification problem with $P(\omega_1) = 0.6, P(\omega_2) = 0.4$, $p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp[-0.5x^2]$ and $p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp[-0.5(x-1)^2]$. Find the decision regions and boundaries for the minimum error rate Bayes classifier.

The decision region $\mathcal{R}_1$ is the set of points x for which $P(\omega_1|x) > P(\omega_2|x)$. The decision region $\mathcal{R}_2$ is the set of points x for which $P(\omega_2|x) > P(\omega_1|x)$. The decision boundary is the set of points x for which $P(\omega_2|x) = P(\omega_1|x)$.

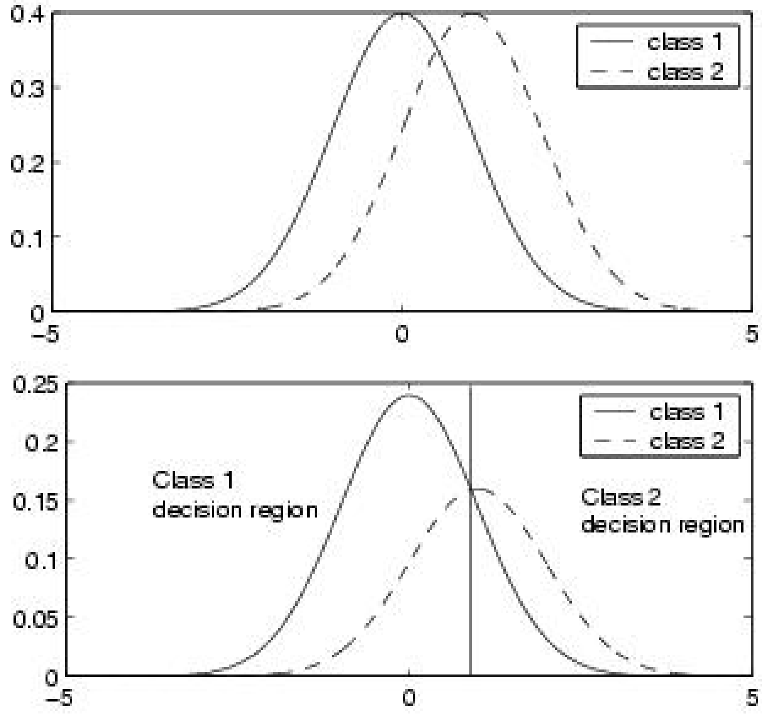

Figure 4.2: Class conditional densities (top) and $P(\omega_1|x)$ and $P(\omega_2|x)$ and decision regions (bottom).

We begin by solving the decision boundary:

$$P(\omega_1|x) = P(\omega_2|x) \Rightarrow p(x|\omega_1)P(\omega_1) = p(x|\omega_2)P(\omega_2),$$

where we used the Bayes formula and multiplied with $p(x)$. Continuing

$$p(x|\omega_1)P(\omega_1) = p(x|\omega_2)P(\omega_2) \Rightarrow \ln[p(x|\omega_1)P(\omega_1)] = \ln[p(x|\omega_2)P(\omega_2)]$$

$$\Rightarrow -x^2/2 + \ln 0.6 = -(x-1)^2/2 + \ln 0.4 \Rightarrow x^2 - 2\ln 0.6 = x^2 - 2x + 1 - 2\ln 0.4.$$

The decision boundary is $x^* = 0.5 + \ln 0.6 - \ln 0.4 \approx 0.91$. The decision regions are (see Figure 4.2)

$$\mathcal{R}_1 = \{x : x < x^*\}, \mathcal{R}_2 = \{x : x > x^*\}.$$

4.6 Discriminant Functions for Normally Distributed Classes

The multivariate normal distribution is the most popular model for class conditional pdfs. There are two principal reasons for this. First is due to its analytical tractability. Secondly, it offers a good model for the case where the feature vectors from a certain class can be thought as noisy versions of some average vector typical to the class. In this section, we study the discriminant functions, decision regions and decision surfaces of the Bayes classifier when the class conditional pdfs are modeled by normal densities. The purpose of this section is to illustrate the concepts introduced and hence special attention should be given to the derivations of the results.

The pdf of the multivariate normal distribution is

$$p_{normal}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\Sigma)}} \exp[-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)], \tag{4.12}$$

where $\mu \in \mathbb{R}^d$ and positive definite $d \times d$ matrix Σ are the parameters of the distribution.

We assume that the class conditional pdfs are normally distributed and prior probabilities can be arbitrary. We denote the parameters for the class conditional pdf for ω_i by μ_i, Σ_i i.e. $p(\mathbf{x}|\omega_i) = p_{normal}(\mathbf{x}|\mu_i, \Sigma_i)$.

We begin with the Bayes classifier defined in Eq. (4.7). This gives the discriminant functions

$$g_i(\mathbf{x}) = p_{normal}(\mathbf{x}|\mu_i, \Sigma_i) P(\omega_i). \tag{4.13}$$

By replacing these with their logarithms (c.f. Theorem 6) and substituting the normal densities, we obtain[2]

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln \det(\Sigma_i) + \ln P(\omega_i). \tag{4.14}$$

We distinguish three distinct cases:

1. $\Sigma_i = \sigma^2 I$, where σ^2 is a scalar and I is the identity matrix;
2. $\Sigma_i = \Sigma$, i.e. all classes have equal covariance matrices;
3. Σ_i is arbitrary.

Case 1

Dropping the constants from the right hand side of Eq. (4.14) yields

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T (\sigma^2 I)^{-1} (\mathbf{x} - \mu_i) + \ln P(\omega_i) = -\frac{||\mathbf{x} - \mu_i||^2}{2\sigma^2} + \ln P(\omega_i), \tag{4.15}$$

The symbol $||\cdot||$ denotes the Euclidean norm, i.e.

$$||\mathbf{x} - \mu_i||^2 = (\mathbf{x} - \mu_i)^T (\mathbf{x} - \mu_i).$$

Expanding the norm yields

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(\mathbf{x}^T\mathbf{x} - 2\mu_i^T\mathbf{x} + \mu_i^T\mu_i) + \ln P(\omega_i). \tag{4.16}$$

[2]Note that the discriminant functions $g_i(\mathbf{x})$ in (4.13) and (4.14) are not same functions. $g_i(\mathbf{x})$ is just a general symbol for the discriminant functions of the same classifier. However, a more prudent notation would be a lot messier.

The term $\mathbf{x}^T\mathbf{x}$ is equal for all classes so it can be dropped. This gives

$$g_i(\mathbf{x}) = \frac{1}{\sigma^2}(\mu_i^T\mathbf{x} - \frac{1}{2}\mu_i^T\mu_i) + \ln P(\omega_i), \tag{4.17}$$

which is a linear discriminant function with

$$\mathbf{w}_i = \frac{\mu_i}{\sigma^2},$$

and

$$w_{i0} = \frac{-\mu_i^T\mu}{2\sigma^2} + P(\omega_i).$$

Decision regions in the case 1 are illustrated in Figure 4.3 which is figure 2.10 from Duda, Hart and Stork.

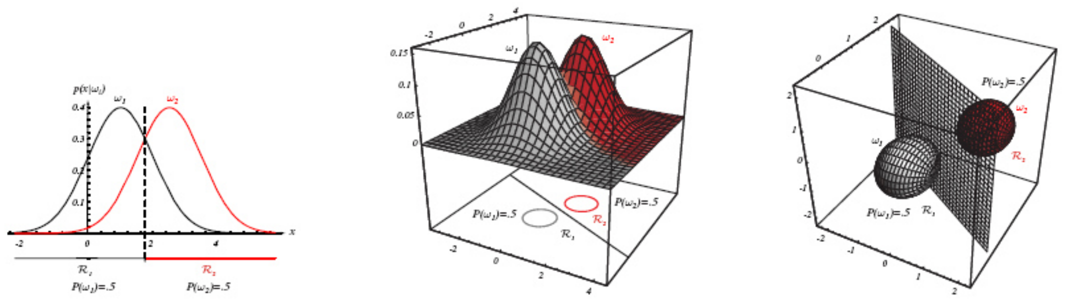

FIGURE 2.10. If the covariance matrices for two distributions are equal and proportional to the identity matrix, then the distributions are spherical in d dimensions, and the boundary is a generalized hyperplane of $d-1$ dimensions, perpendicular to the line separating the means. In these one-, two-, and three-dimensional examples, we indicate $p(\mathbf{x}|\omega_i)$ and the boundaries for the case $P(\omega_1) = P(\omega_2)$. In the three-dimensional case, the grid plane separates $\mathcal{R}_1$ from $\mathcal{R}_2$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Figure 4.3: Case 1

An important special case of the discriminant functions (4.15) is obtained when $P(\omega_i) = \frac{1}{c}$ for all i. Then $\mathbf{x}$ is assigned to the class with the mean vector closest to $\mathbf{x}$:. This is called *minimum distance classifier.* It is a linear classifier as a special case of a linear classifier.

Case 2

We obtain a linear classifier even if the features are dependent, but the covariance matrices for all classes are equal (i.e. $\Sigma_i = \Sigma$). It can be represented by the discriminant functions

$$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{i0}, \tag{4.18}$$

where

$$\mathbf{w}_i = \Sigma^{-1}\mu_i$$

and

$$w_{i0} = -\frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + \ln P(\omega_i).$$

Decision regions in the case 2 are illustrated in Figure 4.4 which is figure 2.11 from Duda, Hart and Stork.

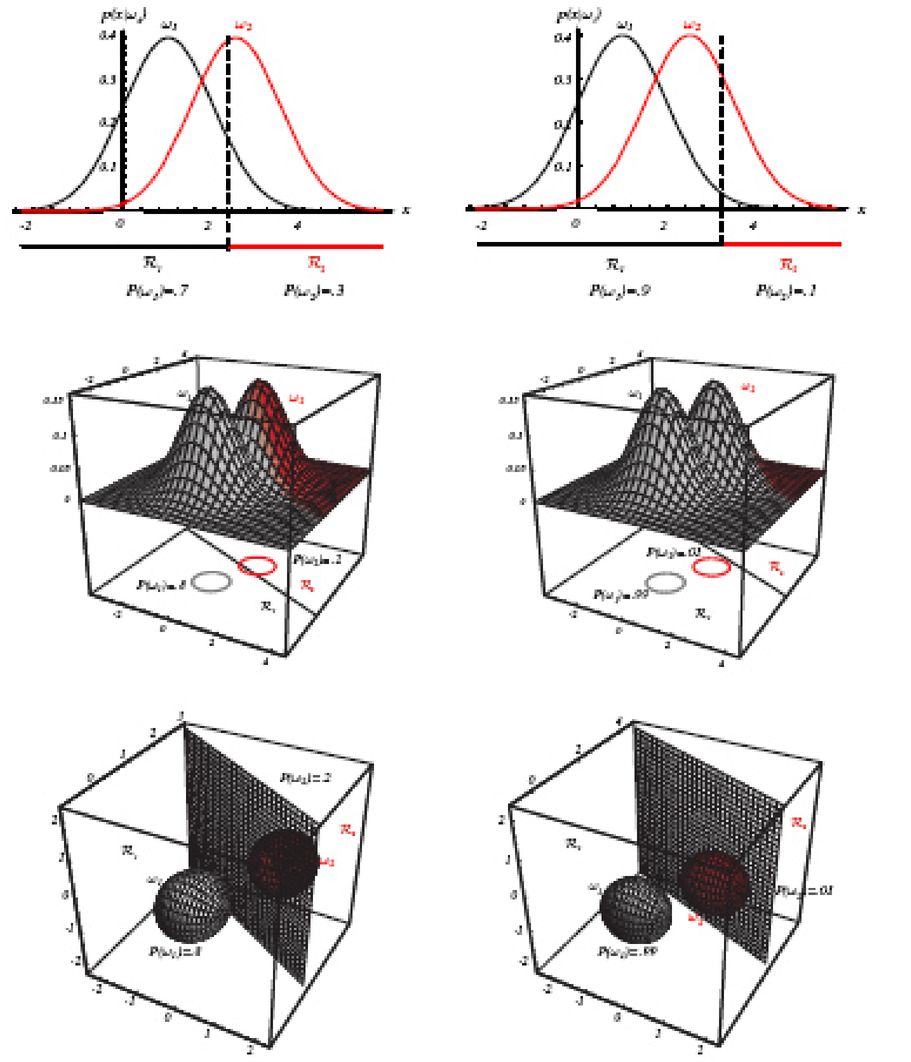

FIGURE 2.11. As the priors are changed, the decision boundary shifts; for sufficiently disparate priors the boundary will not lie between the means of these one-, two- and three-dimensional spherical Gaussian distributions. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Figure 4.4: Case 2

Case 3

Now no additional assumptions about the class conditional pdfs are made. In this case, the discriminant functions

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln\det(\Sigma_i) + \ln P(\omega_i).$$

cannot be simplified much further. Only the constant term $\frac{d}{2}\ln 2\pi$ can be dropped. Discriminant functions are not anymore linear but *quadratic*. They have much more complicated decision regions than the linear classifiers of the two previous cases. Now, decision surfaces are also quadratic and the decision regions do not have to be even connected sets.

Decision regions in the case 3 are illustrated in Figure 4.5 which is Figure 2.14 from Duda, Hart and Stork.

4.7 Independent Binary Features

We consider still the case where the features are binary-valued. That is, their value is either zero (or 'no') or one (or 'yes'). The random variable representing feature vectors is now discrete and so are the class conditional pdfs. The theory of classification is not affected by this. Only, integrals need to be replaced by sums. All the tricks that we have learned this far are applicable.

We make a further assumption that the features are independent and the class conditional marginal density for each feature is the density of the Bernoulli distribution. This yields

$$P(\mathbf{x}|\omega_i) = \prod_{j=1}^{d} q_{ij}^{x_j}(1-q_{ij})^{(1-x_j)}, \tag{4.19}$$

where $q_{ij}, j = 1, \ldots, d$ are parameters for the class conditional density of the class ω_i. The distribution can be interpreted in the way that jth feature answers 'yes' to the question asked with the probability of q_{ij}. (Compare to the coin tossing experiment.) It is still worth emphasizing that the features are independent from each other, i.e. the answer to the jth question does not depend on the answers to the other questions. For a 2-class pattern recognition problem, if $q_{1j} > q_{2j}$ the value of the object's jth feature is 1 more often if the object belongs to the class ω_1.

We now show that the Bayes classifier is linear for this problem. For the Bayes classifier

$$\alpha_{Bayes}(\mathbf{x}) = \arg\max_{\omega_i, i=1,\ldots,c} P(\omega_i|\mathbf{x}),$$

discriminant functions are given as

$$g_i(\mathbf{x}) = P(\mathbf{x}|\omega_i)P(\omega_i).$$

Substituting Eq. (4.19) to the above equation gives

$$g_i(\mathbf{x}) = \prod_{j=1}^{d} q_{ij}^{x_j}(1-q_{ij})^{(1-x_j)}P(\omega_i). \tag{4.20}$$

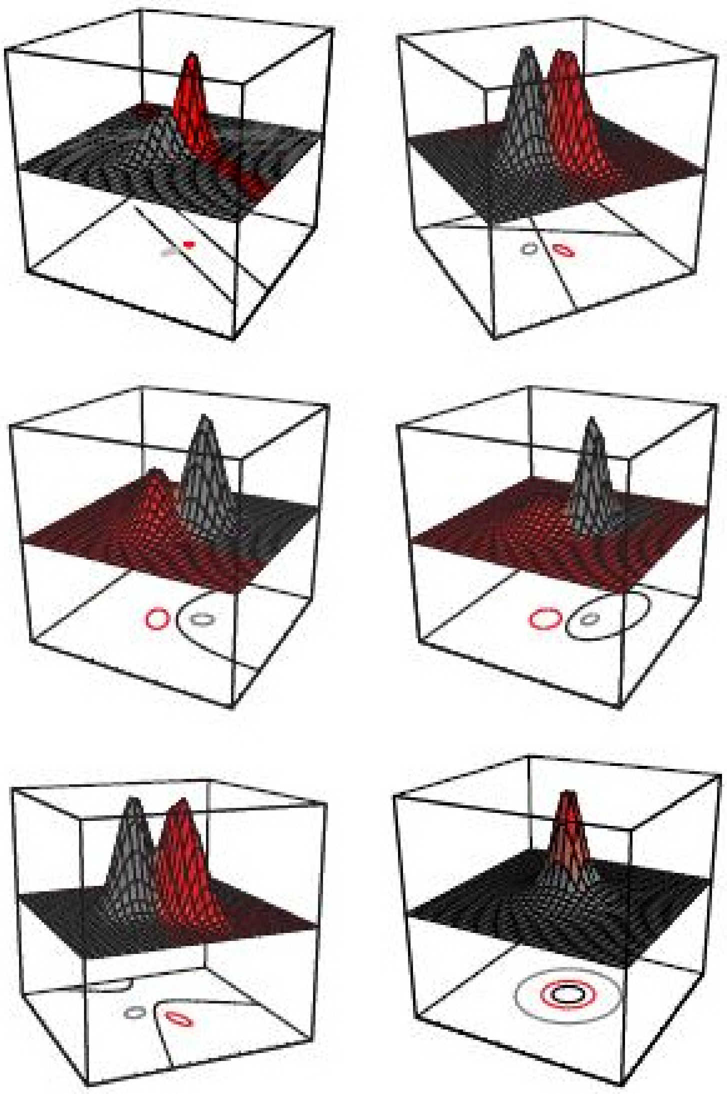

FIGURE 2.14. Arbitrary Gaussian distributions lead to Bayes decision boundaries that are general hyperquadrics. Conversely, given any hyperquadric, one can find two Gaussian distributions whose Bayes decision boundary is that hyperquadric. These variances are indicated by the contours of constant probability density. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Figure 4.5: Case 3

Replacing the right hand side by its logarithm and simplifying give

$$g_i(\mathbf{x}) = \sum_{j=1}^{d}[x_j \ln q_{ij} + \ln(1 - q_{ij}) - x_j \ln(1 - q_{ij})] + \ln P(\omega_i). \tag{4.21}$$

These are linear discriminant functions

$$g_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0},$$

where

$$w_{ij} = \ln \frac{q_{ij}}{1 - q_{ij}},$$

and

$$w_{i0} = \sum_{j=1}^{d} \ln(1 - q_{ij}) + \ln P(\omega_j).$$

In the case of just two classes, the discriminant function is

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) = \sum_{j=1}^{d} \ln \frac{q_{1j}(1 - q_{2j})}{(1 - q_{1j})q_{2j}} x_j + \sum_{j=1}^{d} \ln \frac{(1 - q_{1j})}{(1 - q_{2j})} + \ln \frac{P(\omega_1)}{P(\omega_2)}.$$

We will end this section (and chapter) by studying how the value of the classification error behaves when we add more and more independent features to the classification problem. This is somewhat more pleasant in the discrete case when we do not need to evaluate integrals numerically. However, in order to make computations feasible for large d, we must make some additional simplifying assumptions. Here the results are more important than their derivation and we will keep derivations as brief as possible.

The classification error is

$$E(\alpha_{Bayes}) = 1 - \sum_{\mathbf{x}} \max P(\mathbf{x}|\omega_i)P(\omega_i), \tag{4.22}$$

where $\mathbf{x}$ is summed over all d-bit binary vectors. Again, the greater the probability of the most probable class the smaller the classification error. Consider the 2-class case. We model the classes using Binomial densities, that is $q_{ij} = q_i$ for all features j, and

$$P(\mathbf{x}|\omega_i) = P(\sum x_j = k|q_i) = q_i^k(1 - q_i)^{d-k}, i = 1, 2,$$

where k is the number of 1s in the feature vector $\mathbf{x}$. Let us still assume that the prior probabilities are equal to 0.5 and $q_1 > q_2$. The class of the feature vector is decided by how many 1s it contains. The Bayes classifier is linear and it classifies $\mathbf{x}$ to ω_1 if the number of 1s in $\mathbf{x}$ exceeds a certain threshold t. The threshold t can be computed from (4.21). The classification error is

$$\begin{aligned} E(\alpha_{Bayes}) &= 1 - \sum_{\mathbf{x}:k \le t} 0.5 q_2^k (1 - q_2)^{d-k} + \sum_{\mathbf{x}:k > t} 0.5 q_1^k (1 - q_1)^{d-k} \\ &= 1 - \sum_{k=0}^{t} 0.5 \frac{d!}{k!(d-k)!} q_2^k (1 - q_2)^{d-k} + \sum_{k=t+1}^{d} 0.5 \frac{d!}{k!(d-k)!} q_1^k (1 - q_1)^{d-k}. \end{aligned}$$

Note that the trick here is that we can treat all $\mathbf{x}$ with certain number of 1s (denoted by k) equally. Since there are many fewer different k than there are different $\mathbf{x}$, the computations are much simplified.

The classification error with respect to d when $q_1 = 0.3, q_2 = 0.2$ is plotted in Figure 4.6. Note that the classification error diminishes when the number of features d grows. This is generally true for the Bayes classifiers. However, this does not mean that other, non-optimal classifiers share this property. For example, if we would have (for some reason) selected $t = \lfloor d/2 \rfloor$ (that does not lead to the Bayes classifier), then we would obtain a classification error that would grow towards 0.5 with d. ($\lfloor d/2 \rfloor$ refers to the greatest integer that is lesser than equal to $d/2$.)

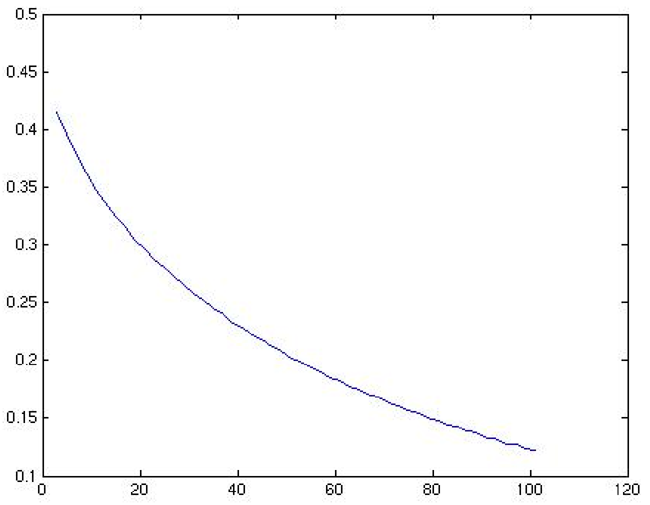

Figure 4.6: Classification error when class conditional pdfs are Binomial.

Chapter 5

Supervised Learning of the Bayes Classifier

5.1 Supervised Learning and the Bayes Classifier

In the previous chapter, we assumed that the class conditional pdfs $p(\mathbf{x}|\omega_i)$ and the prior probabilities $P(\omega_i)$ were known. In practice, this is never the case. Designing a classifier based on the Bayes decision theory implies learning of the class conditional pdfs and priors. In this chapter, we study supervised learning of class conditional pdfs.

For supervised learning we need training samples. As we have defined in Chapter 2, training samples are feature vectors for which the correct classes are known. We refer to the set of training samples as *training data* or as *training set*. In the training set there are feature vectors from each class $\omega_1, \ldots, \omega_c$. We can re-arrange training samples based on their classes: Training samples from the class ω_i are denoted by

$$\mathcal{D}_i = \{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in_i}\},$$

where n_i is the number of training samples from the class ω_i. We assume that the training samples in the sets $\mathcal{D}_i$ are occurrences of the independent random variables. (That is: they were measured from different objects.) Further, these random variables are assumed to be distributed according to $p(\mathbf{x}|\omega_i)$. In short, we say that $\mathcal{D}_i$ is *independent and identically distributed* (i.i.d.).

The training data may be collected in two distinct ways. These are meaningful when we need to learn the prior probabilities. In *mixture sampling*, a set of objects are randomly selected, their feature vectors are computed and then the objects are hand-classified to the most appropriate classes. (This results pairs consisting of a feature vector and the class of the feature vector, but these pairs can be re-arranged to obtain representation as above for the training data.) In *separate sampling*, the training data for each class is collected separately. For the classifier training, the difference of these two sampling techniques is that based on the mixed sampling we can deduce the priors $P(\omega_1), \ldots, P(\omega_c)$ as

$$P(\omega_i) = \frac{n_i}{\sum_{j=1}^{c} n_j}. \tag{5.1}$$

On the other hand, the prior probabilities cannot be deduced based on the separate sampling, and in this case, it is most reasonable to assume that they are known.

The estimation of class conditional pdfs is more difficult. We need now to estimate a function $p(\mathbf{x}|\omega_i)$ based on a finite number of training samples. Next sections are dedicated to different approaches to estimate the class conditional pdfs.

5.2 Parametric Estimation of Pdfs

5.2.1 Idea of Parameter Estimation

We study first an approach where the estimation of the class conditional pdfs $p(\mathbf{x}|\omega_i)$ is reduced to the estimation of the parameter vector θ_i. To do this we must assume that $p(\mathbf{x}|\omega_i)$ belongs to some family of parametric distributions. For example, we can assume that $p(\mathbf{x}|\omega_i)$ is a normal density with unknown parameters $\theta_i = (\mu_i, \Sigma_i)$. Of course, then it is necessary to justify the choice of the parametric family. Good justifications are grounded e.g. on physical properties of the sensing equipment and/or on properties of the feature extractor.

We denote
$$p(\mathbf{x}|\omega_i) = p(\mathbf{x}|\omega_i, \theta_i),$$
where the class ω_i defines the parametric family, and the parameter vector θ_i defines the member of that parametric family. We emphasize that the parametric families do not need to be same for all classes. For example, the class ω_1 can be normally distributed whereas ω_2 is distributed according to the Cauchy distribution. Due to independence of the training samples, the parameter vectors can be estimated separately for each class. The aim is now to estimate the value of the parameter vector θ_i based on the training samples
$$\mathcal{D}_i = \{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in_i}\}.$$

5.2.2 Maximum Likelihood Estimation

We momentarily drop the indices describing the classes to simplify the notation. The aim is to estimate the value of the parameter vector θ based on the training samples $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. We assume that the training samples are occurrences of i.i.d. random variables distributed according to the density $p(\mathbf{x}|\theta)$.

The maximum likelihood estimate or *the ML estimate* $\hat{\theta}$ maximizes the probability of the training data with respect to θ. Due to the i.i.d. assumption, the probability of $\mathcal{D}$ is
$$p(\mathcal{D}|\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i|\theta).$$
Hence
$$\hat{\theta} = \arg\max \prod_{i=1}^{n} p(\mathbf{x}_i|\theta). \tag{5.2}$$

The function $p(\mathcal{D}|\theta)$ - when thought as a function of θ - is termed *the likelihood function.*

Often it is more convenient to work with the logarithm of the likelihood function than with the likelihood function itself. The logarithm is an increasing function and therefore we can safely replace the likelihood function with its logarithm without affecting the maximization result. Hence,

$$\hat{\theta} = \arg\max \sum_{i=1}^{n} \ln p(\mathbf{x}_i|\theta). \tag{5.3}$$

The function $\ln p(\mathcal{D}|\theta) = \sum_{i=1}^{n} \ln p(\mathbf{x}_i|\theta)$ is called the *log-likelihood function.*

5.2.3 Finding ML-estimate

A pleasant feature of the ML-estimate is that we can find it, or at least search for it, numerically. In some important cases, the ML-estimate can be determined analytically. This can be done with a following procedure:

1. Derive the gradient of the likelihood function or the log-likelihood function.
2. Solve the zeros of the gradient and search also all the other critical points of the (log)-likelihood function. (e.g. the points where at least one of the partial derivatives of the function are not defined are critical in addition to the points where gradient is zero.)
3. Evaluate the (log)-likelihood function at the critical points and select the critical point with the highest likelihood value as the estimate.

A word of warning: the ML-estimate does not necessarily exist for all distributions, for example the likelihood function could grow without limit. However, the ML-estimates exist for simple parametric families.

5.2.4 ML-estimates for the Normal Density

We derive the ML-estimates for the normal density

$$p(\mathbf{x}|\theta) = \frac{1}{(2\pi)^{d/2}\sqrt{\det(\Sigma)}} \exp[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)].$$

Consider first the mean vector, i.e. $\theta = \mu$. The covariance matrix Σ is assumed to be known and fixed. (We will see that fixing the covariance will not affect the estimate for mean.) The log-likelihood is

$$l(\theta) = \sum_{i=1}^{n} \ln p(\mathbf{x}_i|\mu) = -\sum_{i=1}^{n} 0.5(\ln[(2\pi)^{d/2}\det(\Sigma)] + (\mathbf{x}_i-\mu)^T\Sigma^{-1}(\mathbf{x}_i-\mu)).$$

Its gradient is

$$\nabla l(\mu) = -\sum_{i=1}^{n} \Sigma^{-1}(\mathbf{x}_i - \mu).$$

Set $\nabla l(\mu) = \mathbf{0}$. The matrix Σ is non-singular, and therefore Σ^{-1} is non-singular. This implies that $\nabla l(\mu) = \mathbf{0}$ has a unique solution. The solution is

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i. \tag{5.4}$$

It is easy to see that this is (the only) local maximum of the log-likelihood function, and therefore it is the ML-estimate. Note that in this case, the ML-estimate is the sample mean of $\mathcal{D}$. As we already hinted, the exact knowledge of the covariance was not necessary for obtaining the ML-estimate for the mean. Hence, the result holds even if both μ and Σ were unknown.

The derivation of the ML-estimate for the covariance matrix is more complicated. Hence, we will skip it and just present the result of the derivation. The ML -estimate is

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T, \tag{5.5}$$

where $\hat{\mu}$ is the ML estimate for μ defined in Eq. (5.4).

A numeric example:

```
 x_1   x_2    x_3     x_4    x_5
 13   -115    119     33    -19
 29    119     -4     17     73

mu =   6.2000
      46.8000

sigma =   5762.6   -3212.2
         -3212.2    1937.0
```

5.2.5 Properties of ML-Estimate

There exist variety of methods for the parameter estimation. Therefore, it is important to know some basic properties of the ML-estimates. It is a favourable property that the ML-estimate can be sought for numerically. In addition, it has some pleasing theoretical properties, which we will briefly describe next.

The ML-estimates are *asymptotically unbiased*. Less formally, this means that they give the correct result on average provided that there are infinitely many training samples available. To be more precise, the expected value of the ML-estimate (over all training sets of size n) is equal to the correct parameter value when n approaches infinity. If n is finite, the above result does not hold in general. For example, the ML-estimate for the covariance matrix of the normal density is biased. When n approaches infinity, the ML-estimate is the best unbiased estimate in the sense that the variance of the ML-estimate is the smallest possible. In addition, the ML-estimate is *asymptotically consistent.* This means that the ML-estimate is arbitrarily close to the true parameter value with the probability of 1 when n tends to infinity.

Almost all optimality properties of the ML-estimate are so called large sample properties. If n is small, little can be said about the ML-estimates in general.

5.2.6 Classification Example

We will illustrate the classifier training based on ML-estimation with Fisher's Iris data. We take the 10 first measurements from each species as training samples. We model the class conditional pdfs by the normal density, and we assume that the priors $P(\omega_i) = \frac{1}{3}$ for all three classes. We represent the training sets for all classes in a matrix form: Each row corresponds to one training sample. For the class ω_1 (iris setosa), the 10 training samples are:

```
    5.1000    3.5000    1.4000    0.2000
    4.9000    3.0000    1.4000    0.2000
    4.7000    3.2000    1.3000    0.2000
    4.6000    3.1000    1.5000    0.2000
    5.0000    3.6000    1.4000    0.2000
    5.4000    3.9000    1.7000    0.4000
    4.6000    3.4000    1.4000    0.3000
    5.0000    3.4000    1.5000    0.2000
    4.4000    2.9000    1.4000    0.2000
    4.9000    3.1000    1.5000    0.1000
```

We obtain based on these:

```
mu1 = 4.8600    3.3100    1.4500    0.2200

Sigma1 =
    0.0764    0.0634    0.0170    0.0078
    0.0634    0.0849    0.0155    0.0148
    0.0170    0.0155    0.0105    0.0040
    0.0078    0.0148    0.0040    0.0056
```

The same for the other classes:

```
mu2 = 6.1000    2.8700    4.3700    1.3800
Sigma2 =
    0.4760    0.1740    0.2910    0.0720
    0.1740    0.1041    0.1191    0.0384
    0.2910    0.1191    0.2141    0.0584
    0.0720    0.0384    0.0584    0.0256

mu3 = 6.5700    2.9400    5.7700    2.0400
Sigma3 =
    0.5821    0.1252    0.4141    0.0782
    0.1252    0.1024    0.1052    0.0794
    0.4141    0.1052    0.3241    0.0752
    0.0782    0.0794    0.0752    0.0764
```

And that is it! We can now classify e.g. the feature vector $\mathbf{x} = [\ 5.9 \quad 4.2 \quad 3.0 \quad 1.5\ ]^T$ by computing the values of the discriminant functions (4.13) for all three classes. The classifier assigns $\mathbf{x}$ to ω_2, or in other words, the flower in the question is iris versicolor.

5.3 Non-Parametric Estimation of Density Functions

It is often the case in pattern recognition that assuming class conditional pdfs to be members of a certain parametric family is not reasonable. The parameter estimation approach of the previous section is then not possible. Instead, we must estimate the class conditional pdfs non-parametrically. In non-parametric estimation (or density estimation), we try to estimate $p(\mathbf{x}|\omega_i)$ in each point $\mathbf{x}$ whereas in parametric estimation we tried to estimate some unknown parameter vector. It is also possible to estimate directly the posterior probabilities $P(\omega_i|\mathbf{x})$ assuming that the training data was collected using mixed sampling.

We formulate the basic problem of density estimation in the following way: We are given the training data $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$,where the samples are i.i.d. , and are all drawn from the unknown density $p(\mathbf{x})$. The aim is to find an estimate $\hat{p}(\mathbf{x})$ for $p(\mathbf{x})$ in every point $\mathbf{x}$.

We will consider two different methods for non-parametric estimation, Parzen windows and k-nearest neighbors method. But before that, we introduce histograms and consider density estimation in a general level.

5.3.1 Histogram

Histograms are the simplest approach to density estimation. The feature space is divided into m equal sized cells or bins $\mathcal{B}_i$. (See Figure 5.1.) Then, the number of the training samples $n_i, i = 1, \ldots n$ falling into each cell is computed. The density estimate is

$$\hat{p}(\mathbf{x}) = \frac{n_i}{Vn}, \quad \text{when} \quad \mathbf{x} \in \mathcal{B}_{\mathrm{i}}, \tag{5.6}$$

where V is the volume of the cell. (All the cells have equal volume and an index is not needed.)

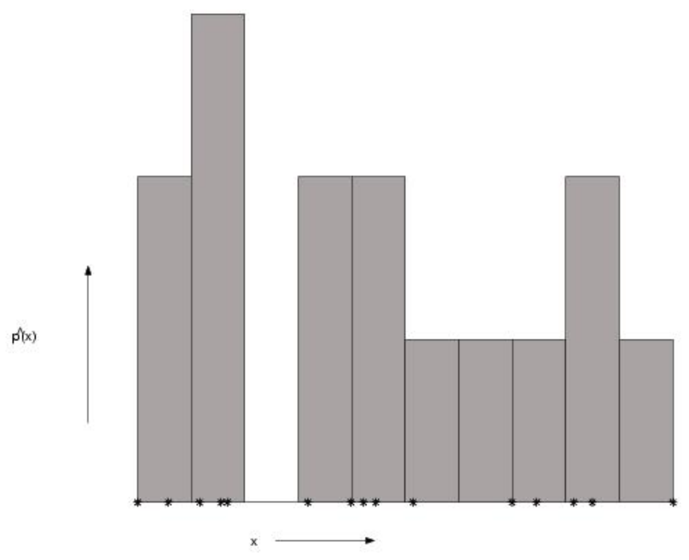

Figure 5.1: Histogram. Aterisks (*) denote the training samples.

The histogram estimate - with a definition as narrow as this one - is not a good way to estimate densities, especially so when there are many (continuous-type) features. Firstly, the density estimates are discontinuous. Secondly, a good size for the cells is hard to select.

5.3.2 General Formulation of Density Estimation

Let us generalize the concept of the histogram by allowing cells of varying shape and size. Particularly, at the proximity of $\mathbf{x}$, we are interested in how to select a suitable cell size/shape producing an accurate $\hat{p}(\mathbf{x})$. Consider the estimate $\hat{p}(\mathbf{x})$ at the point $\mathbf{x}$ and the set (neighborhood) $\mathcal{B}$ surrounding $\mathbf{x}$. The probability that a certain training sample $\mathbf{x}_j$ is in $\mathcal{B}$ is

$$P = \int_{\mathcal{B}} p(\mathbf{x})d\mathbf{x}. \tag{5.7}$$

We still need the probability that k from n training samples fall into the set $\mathcal{B}$. The training samples are independent, and each of them is in the set $\mathcal{B}$ with the probability P. Hence, we can apply the binomial density to the situation. That is, there are exactly k samples in the set $\mathcal{B}$ with the probability

$$P_k = \begin{pmatrix} n \\ k \end{pmatrix} P^k(1-P)^{n-k}, \tag{5.8}$$

where

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{k!(n-k)!}$$

is the number of different subsets of $\mathcal{D}$ of the size k.

Based on the Binomial theorem, we obtain the expected value of k. It is

$$E[k] = \sum_{k=0}^{n} kP_k = \sum_{k=1}^{n} \left(\begin{array}{c} n-1 \\ k-1 \end{array} \right) P^{k-1}(1-P)^{n-k} nP = nP. \tag{5.9}$$

(We have skipped some algebra.) Now, we substitute $E[k]$ by the observed number $\hat{k}$ of training samples falling into $\mathcal{B}$. We obtain an estimate for P:

$$\hat{P} = \hat{k}/n. \tag{5.10}$$

If $p(\mathbf{x})$ is continuous and $\mathcal{B}$ is small enough,

$$\int_{\mathcal{B}} p(\mathbf{x}) d\mathbf{x} \simeq p(\mathbf{x})V,$$

where V is the volume of $\mathcal{B}$. This gives the estimate (or an interpretation for the estimate)

$$\hat{p}(\mathbf{x}) = \frac{\hat{k}}{nV} \simeq \frac{\int_{\mathcal{B}} p(\mathbf{x}) d\mathbf{x}}{\int_{\mathcal{B}} d\mathbf{x}}. \tag{5.11}$$

We can draw the following conclusions:

- The obtained density estimate is a space averaged version of the true density. The smaller the volume V the more accurate the estimate is. However, if n is fixed, diminishing V will lead sooner or later to $\mathcal{B}$ which does not contain any training samples and the density estimate will become useless.

- In more general level two fact need to be accepted. 1) The density estimate is always a space averaged version of the true density. 2) The estimate $\hat{k}$ is only an estimate of nP when n is finite.

The principal question is how to select $\mathcal{B}$ and V. Let us assume that we have unlimited number of training data. We can then approach this problem in the following manner: To estimate $p(\mathbf{x})$, we form a series of cells $\mathcal{B}_1, \mathcal{B}_2, \ldots$ containing $\mathbf{x}$. The cell $\mathcal{B}_1$ is used when we have just one training sample, the cell $\mathcal{B}_1$ is used when we have two training samples and so-forth. The volume of $\mathcal{B}_n$ is denoted by V_n and the number of training samples falling into $\mathcal{B}_n$ is denoted by k_n. The density estimate based on these is

$$p_n(\mathbf{x}) = \frac{k_n}{nV_n}.$$

If we want that

$$\lim_{n\to\infty} p_n(\mathbf{x}) = p(\mathbf{x})$$

then three conditions must be fulfilled

1. $\lim_{n\to\infty} V_n = 0$

2. $\lim_{n\to\infty} k_n = \infty$

3. $\lim_{n\to\infty} k_n/n = 0$.

The conditions may seem rather abstract but they are in fact natural. The first condition assures us that the space averaged $p_n(\mathbf{x})$ will converge to $p(\mathbf{x})$ if there are enough training samples. The second one assures us that the frequency ratio k_n/n will be a good enough estimate for P. The third one states that the number of samples falling to a region $\mathcal{B}_n$ is always a negligibly small portion of the total number of samples. This is required if $p_n(\mathbf{x})$ is to converge at all.

How to guarantee the conditions in the limit of unlimited number of training data? There are two alternatives. We may fix V_n and estimate k_n in the best way possible. Or we can fix k_n and estimate V_n. The precise definition of $\mathcal{B}_n$ is not necessary in either alternatives (though it may be implied.)

The optimality results about the density estimators are necessarily of asymptotic nature. Rosenblatt showed already in 1956 that in the continuous case density estimates based on finite n are necessarily biased.

5.4 Parzen Windows

5.4.1 From Histograms to Parzen Windows

As an introduction to Parzen windows, we consider a certain generalization of the histogram. Assume that the region $\mathcal{B}_n$ is a d-dimensional hypercube. If h_n is the length of the side of the hypercube, its volume is given by $V_n = h_n^d$. We define still a function that receives value 1 inside the hypercube centered at the origin, and value 0 outside the hypercube:

$$\varphi(\mathbf{u}) = \begin{cases} 1 & \text{if} \quad |\mathrm{u_j}| \leq 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

It follows that $\varphi((\mathbf{x}-\mathbf{x}_i)/h_n) = 1$ if $\mathbf{x}_i$ falls in the hypercube of volume V_n centered at $\mathbf{x}$. And $\varphi((\mathbf{x}-\mathbf{x}_i)/h_n) = 0$ otherwise. The number of training samples in this hypercube is therefore given by

$$k_n = \sum_{i=1}^{n} \varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}).$$

This leads to the density estimate

$$p_n(\mathbf{x}) = \frac{k_n}{nV_n} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{V_n}\varphi(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}) \tag{5.13}$$

Above estimate differs from the histogram defined in section 5.3.1 in that we do not define the cells *a-priori.* Rather the cells are defined only after observing $\mathbf{x}$ and they can also intersect each other for different $\mathbf{x}$.

5.4.2 Parzen Estimates

If we consider also other φ functions than above, we obtain a more general approach to density estimation. *The Parzen-window density estimate* at $\mathbf{x}$ using n training samples and *the window function* φ is defined by

$$p_n(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} \varphi(\frac{\mathbf{x} - \mathbf{x}_i}{h_n}). \tag{5.14}$$

The estimate $p_n(\mathbf{x})$ is an average of values of the window function at different points. (See figure 5.2.) Typically the window function has its maximum at the origin and its values become smaller when we move further away from the origin. Then each training sample is contributing to the estimate in accordance with its distance from $\mathbf{x}$. The normal density $\varphi(\mathbf{u}) = \frac{1}{(2\pi)^{d/2}} \exp[-0.5\mathbf{u}^T\mathbf{u}]$ is the most widely applied window function. Usually $V_n = h_n^d$, but the geometric interpretation of V_n as the volume of a hypercube is not completely valid anymore. We will discuss the selection of *window length* h_n in section 5.4.4.

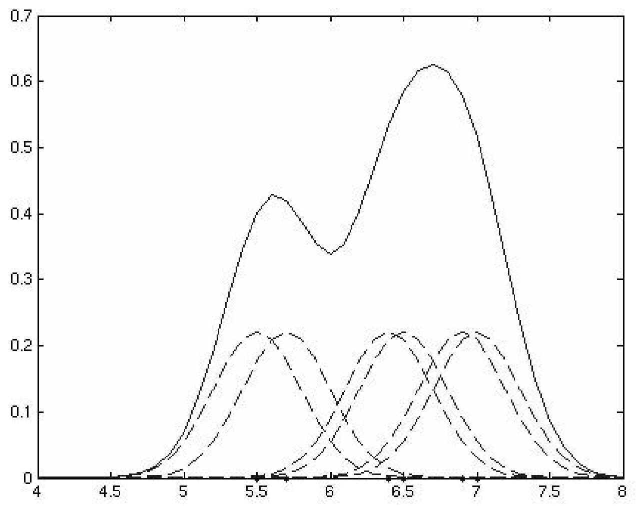

Figure 5.2: Parzen estimate

Sometimes Parzen estimates are called *kernel estimates*. The Parzen estimates were introduced by Emanuel Parzen in 1962. However, Rosenblatt studied similar estimates already in 1956.

5.4.3 Properties of Parzen Estimates

Parzen estimates have several interesting properties, some of which are studied now. For $p_n(\mathbf{x})$ to be a legitimate density, it is required that 1) $p_n(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ and 2) $\int p_n(\mathbf{x})d\mathbf{x} = 1$. If we maintain the relation $h_n^d = V_n$, this is guaranteed if the window function is a legitimate density:

1. $\varphi(\mathbf{x}) \geq 0$ for all $\mathbf{x}$;

2. $\int \varphi(\mathbf{x}) d\mathbf{x} = 1$.

The proof is an exercise.

With an unlimited number of training samples, it is possible to let V_n approach zero, and have $p_n(\mathbf{x})$ to converge to $p(\mathbf{x})$ (for example in the point-wise mean-square sense). For this, it is required that:

1. The density function p must be continuous.

2. The window function must be bounded and a legitimate density.

3. The values of the window function must be negligible at infinity.

4. $V_n \to 0$ when $n \to \infty$.

5. $nV_n \to \infty$ when $n \to \infty$.

In addition to the assumptions about the window function and the window length, we needed to make assumptions about the true density. The requirements concerning the window function are natural and e.g. the normal density is in accordance with them.

For pattern recognition, these optimality/convergence properties are important, because with these properties we are able to show that the classification error of a classifier based on Parzen windows tends to the Bayes error when n approaches infinity. This holds also more generally: When the density estimates converge (in a certain sense) to the true class conditional pdfs, and the prior probabilities are properly selected, then the error of the resulting classifier tends to the Bayes error. We do not go into detail with the conditions for the convergence, since their proper treatment would require too much effort.

5.4.4 Window Width

The window width satisfying the requirements in the previous paragraph can be chosen as

$$h_n = h_1/\sqrt{n}.$$

In practice, we need to select h_1. This is not easy. The choice often is application specific and it is rooted in the properties of the density estimate required by the application. If h_1 (h_n) is too large, then the density estimate $p_n(\mathbf{x})$ will be very smooth and 'out-of-focus'. If h_1 is too small, then the estimate $p_n(\mathbf{x})$ will be just superposition of n sharp pulses centered at training samples, i.e. an erratic and noisy estimate of the true density.

5.4.5 Parzen Classifiers

It is straight-forward to apply Parzen estimates to the classification problems. For each class we have the training samples $\mathcal{D}_1, \ldots, \mathcal{D}_n$. Then, if possible, we first estimate the priors $P(\omega_1), \ldots, P(\omega_c)$. Thereafter, we estimate the class conditional pdfs $p(\cdot|\omega_i)$ based on Parzen windows, and we construct the Bayes classifier based on the estimates. The hard part is that it is not possible to estimate $p(\cdot|\omega_i)$ for all possible feature vectors $\mathbf{x}$. For each *test point* $\mathbf{x}$ to be classified, we must first compute the Parzen estimate of the each class conditional density, and only after that we can classify the test point. The Parzen-classifier can be written simply as

$$\alpha_{Parzen}(\mathbf{x}) = \arg \max_{\omega_i, i=1,\ldots,c} \hat{P}(\omega_i) \frac{1}{n_i} \sum_{j=1}^{n_i} \frac{1}{h_i^d} \varphi(\frac{\mathbf{x} - \mathbf{x}_{i_j}}{h_i}), \tag{5.15}$$

where $\hat{P}(\omega_i)$ are the estimates for prior probabilities, $\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in_i}$ are training samples from ω_i, φ is the window function (usually the same window function is used for all classes), and h_i is the window width for the class ω_i.

As can be seen based on Eq. (5.15), Parzen classifier demand much more computation than the classifiers based on the ML-estimation. Every classification with Parzen classifiers requires n evaluations of a pdf, where n is the total number of training samples.

Note that by substituting $\hat{P}(\omega_i) = n_i/n$ in Eq. (5.15), the Parzen classifier can be rewritten as

$$\alpha_{Parzen}(\mathbf{x}) = \arg \max_{\omega_i, i=1,\ldots,c} \frac{1}{n} \sum_{j=1}^{n_i} \frac{1}{h_i^d} \varphi(\frac{\mathbf{x} - \mathbf{x}_{i_j}}{h_i}).$$

5.4.6 Probabilistic Neural Networks

Parzen classification can be also implemented in an neural network configuration. Such implementation is termed *Probabilistic neural network (PNN)*. Then it is assumed that the training samples have been normalized to the unit length, i.e.

$$||\mathbf{x}_{i_j}|| = 1.$$

Also the test point has to be normalized similarly. This is not reasonable in all applications. We denote the normalized training samples by $\mathbf{y}_i$. We further assume that the window function is the normal density.

A PNN consists of d input units comprising the input layer, n pattern units comprising of the pattern layer, and c category units. Each input unit is connected to each pattern unit. There is a pattern unit for each training sample. Each pattern unit is connected to one and only one category unit corresponding the category of the training sample. For the normalized test point $\mathbf{x}$, each pattern unit computes the inner product

$$net_i = \mathbf{y}_i^T \mathbf{x}$$

and feeds the value $\exp[(net_i - 1)/h_n^2]$ to the category unit to which it is connected to. Each output unit sums up the contributions of the pattern units connected to it and assigns $\mathbf{x}$ to the class with the greatest sum. This indeed implements the Parzen classifier because

$$\begin{aligned}\varphi(\frac{\mathbf{x}-\mathbf{y}_i}{h_n}) &\propto \exp[-(\mathbf{x}-\mathbf{y}_i)^T(\mathbf{x}-\mathbf{y}_i)/(2h_n^2)] && (5.16)\\ &= \exp[-(\mathbf{x}^T\mathbf{x}+\mathbf{y}_i^T\mathbf{y}_i - 2\mathbf{x}^T\mathbf{y}_i)/(2h_n^2)] && (5.17)\\ &= \exp[(net_i-1)/h_n^2], && (5.18)\end{aligned}$$

where the last equality is due to our normalization assumptions. Notation $a(x) \propto b(x)$ means that $a(x) = C \cdot b(x)$ for some constant C and for all x.

5.5 k-Nearest Neighbors Classifier

5.5.1 k_n Nearest Neighbors Density Estimation

With the Parzen classification, the design of the classifier involved selecting window functions and suitable window widths. This is not an easy task. One possibility is let them depend on the training data. With respect to the general formulation of the density estimation, this means fixing k_n and computing of suitable (small enough) V_n based on the selected k_n. To be more precise, we place the center of the cell $\mathcal{B}_n$ into the test point $\mathbf{x}$ and let the cell grow until it encircles k_n training samples to estimate $p(\mathbf{x})$. These k_n training samples are k_n nearest neighbors of $\mathbf{x}$. Here, k_n is a given parameter. The k_n *nearest neighbor (KNN) density estimate* is given by

$$p_n(\mathbf{x}) = \frac{k_n}{nV_n}, \tag{5.19}$$

where V_n is the volume of the smallest possible $\mathbf{x}$ centered ball that contains k_n training samples, and n is the total number of training samples. If the true density is high in the proximity of $\mathbf{x}$, the cell $\mathcal{B}_n$ is small and the density estimate is accurate. If the true density is low in the proximity of $\mathbf{x}$, the cell $\mathcal{B}_n$ is large and the density estimate is not that accurate.

Let $p(\mathbf{x})$ be continuous at $\mathbf{x}$. The KNN density estimate converges if

1. $k_n \to \infty$ when $n \to \infty$;
2. $k_n/n \to 0$ when $n \to \infty$.

A surprising property of the KNN density estimate is that although it is continuous, its partial derivatives are necessarily not continuous. Typically, KNN density estimates are quite 'spiky' and the density estimates by Parzen windows more smooth.

It is also possible to estimate posterior probabilities $P(\omega_i|\mathbf{x})$ directly with the KNN method. This leads to more interesting result than in the case of Parzen windows. We assume that the training samples $\mathcal{D}_1, \ldots, \mathcal{D}_c$ have been collected by the mixture sampling. Also, we drop the indices describing the number of training samples from all estimates and parameters. This is to avoid confusion in notation. Now, center a cell encircling k training samples at the test point $\mathbf{x}$. The k training samples can be from any class. k_i of these belongs to the class ω_i. Then, the estimate for the joint probability of $\mathbf{x}$ and ω_i is

$$p(\mathbf{x}, \omega_i) = \frac{k_i}{nV}.$$

Using the definition of the conditional densities, we obtain an estimate for the posterior probability

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}, \omega_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x}, \omega_i)}{\sum_{j=1}^{c} p(\mathbf{x}, \omega_j)} = \frac{k_i}{\sum_{j=1}^{c} k_j} = \frac{k_i}{k}. \tag{5.20}$$

This posterior probability estimate is maximal for the class with the highest k_i and $\mathbf{x}$ is assigned to that class. This is *the k nearest neighbor (KNN) classification rule.*

5.5.2 Nearest Neighbor Rule

We consider first *the nearest neighbor (NN) classification*, where $\mathbf{x}$ is assigned to the class of its nearest neighbor. That is, $k = 1$. We can write this rule also as a traditional function:

$$\alpha_{nn}(\mathbf{x}) = \arg \min_{\omega_i, i=1,\ldots,c} \min_{j} ||\mathbf{x} - \mathbf{x}_{i_j}||. \tag{5.21}$$

It was required that $k_n \to \infty$ when $n \to \infty$ in order to KNN density estimates to converge to the true density function. With the NN rule, $k_n = 1$ for all n. Therefore, the NN classification rule is not optimal. However, we can show that the error of the NN classifier is at most twice the Bayes error when the number of training samples tends to infinity. More precisely,

$$E(\alpha_{bayes}) \leq E(\alpha_{nn}) \leq E(\alpha_{bayes})(2 - \frac{c}{c-1} E(\alpha_{bayes})).$$

The upper bound is reached when the true class conditional pdfs are identical for all classes.

The rather positive error bound above is a theoretical one. In real life, we do not have infinite number of training samples and there are no corresponding results for the case when we have a finite number of training samples. However, the NN classifier is very simple, and in practice it often works well. We explain this heuristically. Denote the nearest neighbor of the test point $\mathbf{x}$ by $\mathbf{x}^*$. Let the class of $\mathbf{x}^*$ be ω^*. Because we assume that the training samples have been correctly classified, ω^* maximizes the posterior $P(\omega_i|\mathbf{x}^*)$ with respect to classes $\omega_i, i = 1, \ldots, c$ with high probability. Then, if $\mathbf{x}$ is close to $\mathbf{x}^*$, ω^* maximizes also the posterior $P(\omega_i|\mathbf{x})$ with respect to classes $\omega_i, i = 1, \ldots, c$ with high probability if the posteriors are smooth enough.

We finally consider the geometry of the decision regions of the NN rule. The decision region for class ω_i is formed by the cells whose points are closer to some training sample in $\mathcal{D}_i$ than any other training sample. This kind of partition of the feature space is termed *Voronoi diagram.* See Figure 5.3 due to Duda, Hart and Stork.

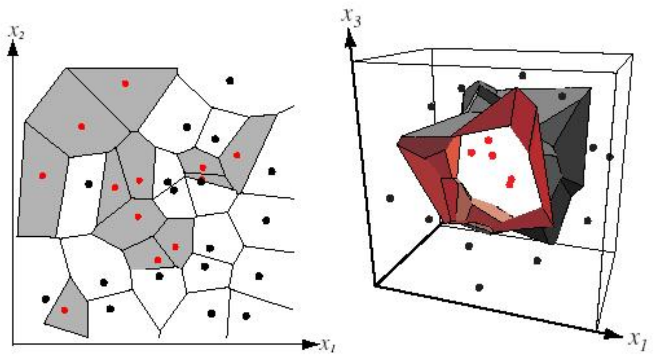

FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Figure 5.3: Decision regions for the nearest neighbor rule

5.5.3 k Nearest Neighbor Classification Rule

The KNN classification rule consists of finding k-nearest training samples to the test point $\mathbf{x}$ and classifying $\mathbf{x}$ to the class which is most frequent in those k nearest neighbors of $\mathbf{x}$. Note that the KNN rule is - as all decision rules are - a function from the feature space to the set of classes, although it is not easy to write the KNN rule as a 'traditional' function.

In the two class case, we following error estimate holds in the case of unlimited number of training samples:

$$E(\alpha_{bayes}) \leq E(\alpha_{knn}) \leq E(\alpha_{bayes}) + \sqrt{\frac{2E(\alpha_{nn})}{k}}.$$

Also, from this error estimate it follows that the KNN rule is optimal when $k \to \infty$. In real life, the situation is once again different: Especially when the feature space is high-dimensional, the required number of training samples grows very rapidly with k.

The KNN classifier and the NN classifier suffer from the same practical problem as the Parzen classifier. The computational complexity of classification grows linearly with the number of training samples, because the distance from each training sample to the test point must be computed before the classification is possible. This may be a problem if the classifier needs to be very fast. Let us note, however, that there exist several techniques to speed up KNN classifiers.

5.5.4 Metrics

We have assumed until now that the distance between the points **a** and **b** in the feature space (feature vectors) is measured with the Euclidean metric:

$$L(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}|| = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}.$$

However, many other distance measures or *metrics* can be applied. The selection of a metric obviously affects the KNN classification results. Especially, the scaling of the features has a fundamental effect to the classification results although this transformation merely accounts to a different choice of units for features.

Formally, a metric is a function $L(\cdot, \cdot)$ from $\mathbb{R}^d \times \mathbb{R}^d$ to $\mathbb{R}$. For all vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$, a metric L must satisfy

1. $L(\mathbf{a}, \mathbf{b}) \geq 0$
2. $L(\mathbf{a}, \mathbf{b}) = 0$ if and only if $\mathbf{a} = \mathbf{b}$.
3. $L(\mathbf{a}, \mathbf{b}) = L(\mathbf{b}, \mathbf{a})$
4. $L(\mathbf{a}, \mathbf{b}) + L(\mathbf{b}, \mathbf{c}) \geq L(\mathbf{a}, \mathbf{c})$.

The only one of these properties which we occasionally want to sacrifice is the property 2. Other properties are important for pattern recognition applications.

Some examples of useful metrics:

- Minkowski metrics:

$$L_m(\mathbf{a}, \mathbf{b}) = (\sum_{i=1}^{d} (|a_i - b_i|^m))^{1/m}.$$

- L-infinity metric

$$L_\infty(\mathbf{a}, \mathbf{b}) = \max_i |a_i - b_i|.$$

- Mahalanobis-distance

$$L_{Mahalanobis,C}(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a} - \mathbf{b})^T C^{-1} (\mathbf{a} - \mathbf{b})},$$

  where C is a given positive definite matrix.

5.6 On Sources of Error

We have mentioned repeatedly in this chapter that the optimality properties of the classifiers hold only if we have an infinite number of training samples. However, we always have only a finite number of them. In this section, we discuss the sources of errors when the number of training samples is finite.

Before discussing error sources, it is important to note that in pattern recognition two error types can be distinguished: *the training error* and *the test error*. *The training error* refers to the classification errors for the training samples, i.e., how large portion of the training samples is misclassified by the designed classifier. Here, we are *n*ot concerned with this error type. The important error type is the *the test error*, which describes how large portion of all possible objects are misclassified, and the discussions below refer to the test error. Note that the Bayes classifier minimizes the test error when the assumptions made about class conditional pdfs and priors hold. The estimation of the test error is a challenging task, and we will return to the issue in Chapter 7. In particular, the training error, in most cases, underestimates the test error.

Before studying the sources of the test error, we recall what kind of inputs different classifiers required for their design. First, all supervised classifiers need training samples. Second, the classifiers based on the ML-estimation need knowledge about the parametric families of the class conditional pdfs. Parzen classifiers need to be given the window functions and window lengths. For KNN classifiers, we need to select the parameter k.

The Bayes error is a necessary error source of every pattern classification problem. This error can never be reduced without changing the classification problem. If we take into account the whole pattern recognition system, we can change the classification problem e.g. by adding an extra feature. This can possibly reduce the Bayes error.

The second source of classification error with practical classifiers is that the probabilistic model for the problem is incorrect. This is termed *model error*. In ML-based classification, these are due to having incorrect (parametric) model for class conditional densities, i.e. the data from the class ω_i is not distributed according to any of the densities $p(\cdot|\omega_i, \theta_i)$. In the non-parametric case, we interpret a bad selection of window functions, window lengths, or the parameter k as model errors. The model error is more important with the parametric ML-based classifiers than with the non-parametric classifiers. This type of error can be reduced by selection a better probabilistic model for the problem. Again, this is rarely straight-forward.

The third and final error source is the *estimation error*. This error is due having only a finite number of training samples. This error is more pronounced in the non-parametric case where more training samples are required than with ML-based classifiers. In the ML-estimation, the larger the parametric family the lesser the model error but the greater the estimation error, i.e. there is a trade-off between the model and estimation errors. The estimation error can be reduced by adding more training samples.

In general, if we add features to the pattern recognition system, we also increase the estimation error. Further, the required number of training samples grows exponentially with the number of features in the non-parametric case. Hence, adding one more feature to the pattern recognition system might not always be a good idea. The phenomenon is often termed the *curse of dimensionality* (the term is due to Bellman) that refers to the exponential growth of hypervolume as a function of dimensionality.

Chapter 6

Linear Discriminant Functions and Classifiers

6.1 Introduction

To this point, we have designed classifiers by estimating the class conditional pdfs and prior probabilities based on training data. Based on the estimated class conditional pdfs and prior probabilities, we have then derived the Bayes classifier. This way the classifier minimizes the classification error when the estimated pdfs and priors are close to the true ones. The problem was that the estimation problems are challenging, and with the finite number of training samples, it is impossible to guarantee the quality of estimates.

In this Chapter, we consider another way to derive the classifier. We assume that we know the parametric forms of the discriminant functions in a similar way as with the ML estimation. The estimation will be formulated as a problem of minimizing a criterion function - again much in a same way than with the ML technique. However, now we are aiming directly to find the discriminant functions without first estimating the class conditional pdfs. An example illustrates positive features of this type of classifier design: Consider a two-category classification problem, where the class conditional pdfs are normal densities with equal covariances. Then, as stated in Section 4.6, the Bayes classifier is linear, and it can be represented with the following discriminant function:

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0,$$

where

$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2)$$

and

$$w_0 = -\frac{1}{2}(\mu_1 - \mu_2)^T\Sigma^{-1}(\mu_1 - \mu_2) + \ln P(\omega_1) - \ln P(\omega_2).$$

For deriving this discriminant function, it was necessary to estimate $2d+(d^2+d)/2+1$ parameter values. The first term is due to the two mean vectors, the second is due to the single (symmetric) covariance matrix, and the third one is due to the priors which sum to one. Hence, it might be reasonable to estimate directly the discriminant function, which has only $d + 1$ parameter values: d of them from the *weight vector* $\mathbf{w}$, and one of them from *the threshold weight* w_0. In addition, the parameterization of the discriminant functions may sometimes be more natural than the parameterization of the class conditional pdfs.

The estimation of $\mathbf{w}$ and w_0 will be performed by minimizing a criterion function as it was already mentioned. The criterion can be e.g. the number of misclassified training samples but more often it is some related function. Note, however, that in this case we will lose the direct contact with the test-error which is much more important quantity than the training error.

6.2 Properties of Linear Classifiers

6.2.1 Linear Classifiers

Recall first how the classifier was defined with the help of discriminant functions. For each class, we had a discriminant function $g_i, i = 1, \ldots, c$ whose input was a feature vector $\mathbf{x}$. The feature vector $\mathbf{x}$ was assigned to ω_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x})$$

for all $i \neq j$. (If $g_i(\mathbf{x}) = g_j(\mathbf{x})$ we can use some arbitrary rule such as classifying $\mathbf{x}$ to ω_i if $i < j$. As with the Bayes classifier, the selection of such a rule bears no importance.)

A discriminant function is said to be *linear* if it can written as

$$g_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0},$$

where $\mathbf{w}_i = [w_{i1}, \ldots, w_{id}]^T$ is the weight vector and the scalar w_{i0} is threshold weight. The classifier relying only on the linear discriminant functions is called linear classifier or linear machine.

6.2.2 The Two Class Case

Consider a 2-class classification problem and a linear classifier. The points on the decision surface satisfy

$$g_1(\mathbf{x}) = \mathbf{w}_1^T \mathbf{x} + w_{10} = \mathbf{w}_2^T \mathbf{x} + w_{20} = g_2(\mathbf{x}).$$

From this, we get

$$(\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{x} + w_{10} - w_{20} = 0.$$

Denote $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_2$ and $w_0 = w_{10} - w_{20}$, and represent the classifier with a single discriminant function. We obtain the decision surface

$$\mathcal{R}_{12} = \{\mathbf{x} | \mathbf{w}^T \mathbf{x} + w_0 = 0\}. \tag{6.1}$$

This is a hyper-plane. A hyper-plane is a point when $d = 1$, a line when $d = 2$, and so-forth.

We choose two arbitrary points laying on the decision surface, say $\mathbf{x}_1$ and $\mathbf{x}_2$. Now

$$\mathbf{w}^T\mathbf{x}_1 - \mathbf{w}^T\mathbf{x}_2 - w_0 + w_0 = \mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0.$$

Hence, the weight vector $\mathbf{w}$ is normal to the decision surface. We can write

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||},$$

where $\mathbf{x}_p$ is the normal projection of the feature vector $\mathbf{x}$ to the decision surface $\mathcal{R}_{12}$. Then, because $g(\mathbf{x}_p) = 0$, we get

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = \mathbf{w}^T(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}) + w_0 = r||\mathbf{w}||.$$

The scalar r is greater than zero if $\mathbf{x}$ belongs to ω_1 and less than zero if $\mathbf{x}$ belongs to ω_2. Also, a high absolute value of $g(\mathbf{x})$ means that $\mathbf{x}$ lies far away from the decision surface. Figure 6.1 clarifies. It is Fig. 5.2 from Duda, Hart and Stork.

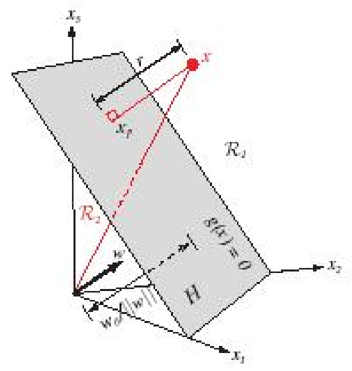

FIGURE 5.2. The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Figure 6.1: Geometry of linear 2-class classifier

6.2.3 The c Class Case

The results of the previous paragraph obviously hold if we consider just 2 of the c classes. When we take all classes into account, we note that the boundary between $\mathcal{R}_i$ and $\mathcal{R}_j$ is a portion of the hyper-plane

$$H_{ij} = \{\mathbf{x}|g_i(\mathbf{x}) = g_j(\mathbf{x})\}.$$

Moreover, we can characterize also decision regions. Namely, these are *convex.*

A set $R \subset \mathbb{R}^d$ is convex (Figure 6.2), if for all $\mathbf{r}_1, \mathbf{r}_2 \in R$, also $(1-\lambda)\mathbf{r}_1 + \lambda\mathbf{r}_2 \in R$,when $\lambda \in [0,1]$. In other words, if R is convex each line segment between its points belongs in whole to R. For example, a ball is convex set in the 3-dimensional Euclidean space. Convex sets are also connected, and therefore a decision region of a linear classifier is connected.

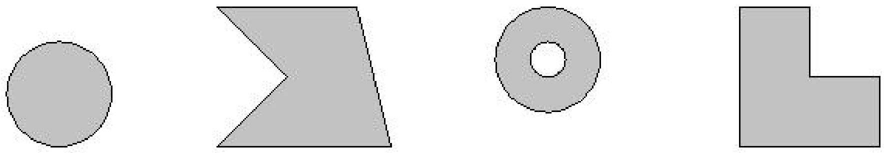
Figure 6.2: Only the left-most of above sets is convex and the others are not convex.

To prove the convexity of the decision regions of linear classifiers, consider two points $\mathbf{r}_1, \mathbf{r}_2$ belonging to the decision region $\mathcal{R}_i$. Because these points belong to $\mathcal{R}_i$, it holds that

$$g_i(\mathbf{r}_1) > g_j(\mathbf{r}_1), g_i(\mathbf{r}_2) > g_j(\mathbf{r}_2) \tag{6.2}$$

for all $j \neq i$. Now, consider the point $(1-\lambda)\mathbf{r}_1 + \lambda\mathbf{r}_2$, where $\lambda \in [0,1]$. Due to linearity of the discriminant functions

$$g_i((1-\lambda)\mathbf{r}_1 + \lambda\mathbf{r}_2) = \mathbf{w}_i^T((1-\lambda)\mathbf{r}_1 + \lambda\mathbf{r}_2) + w_{i0} > g_j((1-\lambda)\mathbf{r}_1 + \lambda\mathbf{r}_2). \tag{6.3}$$

This means that the decision regions are convex. It is an exercise to prove that (6.3) follows from (6.2).

The convexity of the decision regions naturally limits the number of classification tasks which can effectively be solved by linear machines.

6.3 Linearly Separable Training Samples

Consider first the 2-class case. We assume that the classification task is such that the classification error is very small for some linear classifier. We have training samples from the classes ω_1 and ω_2. We denote these as before

$$\mathcal{D}_i = \{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in_i}\}, i = 1, 2,$$

where n_i is the number of samples belonging to ω_i. Now, if a such linear classifier exists that it classifies all the training samples correctly, i.e.

$$g(\mathbf{x}_{1j}) > 0,$$

for all $j = 1, \ldots, n_1$ and

$$g(\mathbf{x}_{2j}) < 0$$

for all $j = 1, \ldots, n_2$, we say that the training sets/samples $\mathcal{D}_1$ and $\mathcal{D}_2$ are *linearly separable.*

Two-class linear classifiers and the above condition can be presented in a more compact form if we make two modifications to the notation:

1) We form the *augmented feature vector* $\mathbf{y}$ based on $\mathbf{x}$:

$$\mathbf{y} = [1, x_1, \ldots, x_d]^T.$$

Similarly, the *augmented weight vector* is obtained from $\mathbf{w}$ and w_0:

$$\mathbf{a} = [w_0, w_1, \ldots, w_d]^T = [w_0, \mathbf{w}^T]^T.$$

Linear discriminant functions can now be written as

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = \mathbf{a}^T\mathbf{y} = g(\mathbf{y}).$$

2) We denote by $\mathbf{y}_{ij}$ the augmented feature vector generated from the training sample $\mathbf{x}_{ij}$. In the 2-class case, we reduce the two training sets $\mathcal{D}_1, \mathcal{D}_2$ to a single training set by simply replacing the training samples from ω_2 by their negatives. This because

$$\mathbf{a}^T\mathbf{y}_{2j} < 0 \Leftrightarrow \mathbf{a}^T(-\mathbf{y}_{2j}) > 0.$$

Hence, we can forget about ω_2, replace its training samples by their negatives, and attach them to the training set for ω_1. We denote the resulting training set by $\mathcal{D} = \{\mathbf{y}_1, \ldots, \mathbf{y}_{n_1+n_2}\}$. Note that 'replacement by negatives' must be performed expressly for *augmented* feature vectors. (Why?).

Example. Consider the training samples

$$\mathcal{D}_1 = \{[1,1]^T, [2,2]^T, [2,1]^T\}$$

and

$$\mathcal{D}_2 = \{[1,-1]^T, [1,-2]^T, [2,-2]^T]$$

The modified training set is then

$$\mathcal{D} = \{[1,1,1]^T, [1,2,2]^T, [1,2,1]^T, [-1,-1,1]^T, [-1,-1,2]^T, [-1,-2,2]^T\}.$$

Given that these two modifications have been performed, we can re-define linear separability: The training set $\mathcal{D}$ is linearly separable if there exists such augmented weight vector $\mathbf{a}$ that

$$\mathbf{a}^T\mathbf{y}_j > 0 \tag{6.4}$$

for all $\mathbf{y}_1, \ldots, \mathbf{y}_{n_1+n_2}$.

If we assume that the training samples are linearly separable, then we can try to solve the system of inequalities (6.4) to design the linear classifier. One possible solution method is introduced in the next section. However, a solution - if exists - is not unique. 1) If we have a solution vector $\mathbf{a}$ which is a solution to the system (6.4), then we can always multiply this vector a positive constant to obtain another solution. (Remember that the weight vectors were normals to the decision surfaces). 2) It is possible to have multiple decision surfaces that separate the training sets. Then, we have many (genuinely different) augmented weight vectors that solve the system (6.4).

6.4 Perceptron Criterion and Algorithm in 2-Class Case

6.4.1 Perceptron Criterion

To find a weight vector $\mathbf{a}$ that satisfies

$$\mathbf{a}^T\mathbf{y}_j > 0$$

for all j, we formulate an optimization problem. We search the minimum argument of the function

$$J_p(\mathbf{a}) = \sum_{j:\mathbf{a}^T\mathbf{y}_j \leq 0} -\mathbf{a}^T\mathbf{y}_j. \tag{6.5}$$

The function J_p is termed *Perceptron criterion.* In other words, we sum the inner products of the weight vector $\mathbf{a}$ and those training samples that are incorrectly classified. The Perceptron criterion is the negative of that sum. Above, we have (implicitly) agreed that augmented feature vectors $\mathbf{y}_j$ for which $\mathbf{a}^T\mathbf{y}_j = 0$ are incorrectly classified. This assumption is useful in preventing degenerate solutions $\mathbf{a} = \mathbf{0}$[1].

The value $J_p(\mathbf{a})$ is always greater than equal to zero. The Perceptron criterion obtains its minimum (that is zero) when $\mathbf{a}$ solves the system of inequalities (6.4). (There is also a degenerate case. When $\mathbf{a} = \mathbf{0}$, $J_p(\mathbf{a}) = 0$. However, this obviously is not the solution we are searching for.) To sum up, we have converted the problem of solving inequalities into an optimization problem.

The criterion $\sum_{j:\mathbf{a}^T\mathbf{y}_j \leq 0} -1$ would also fulfill the above requirements. However, this criterion function is not continuous which would lead to numerical difficulties. Instead, J_p is continuous.

6.4.2 Perceptron Algorithm

Usually, the analytic (i.e. by paper and pen) minimization of the criterion J_p is not possible. Therefore, we must rely on numerical techniques to solve the minimization problem. The most simple method for the numerical minimization is the *steepest*

[1]Note that this assumption/agreement has no effect on the value of the Perceptron criterion. However, it will carry importance when defining the gradient of the Perceptron criterion needed for deriving the Perceptron algorithm

descent method, which is well applicable here. In the steepest descent method, we choose an initial value (denoted by $\mathbf{a}(0)$) for $\mathbf{a}$ and start updating $\mathbf{a}$ little by little to the direction of the negative gradient of J_p at $\mathbf{a}(t)$ at each iteration $t = 0, 1, 2, \ldots$. The idea behind the steepest descent method is the value of any continuous function f at $\mathbf{x}$ decreases most rapidly to the direction of its negative gradient $-\nabla f(\mathbf{x})$. The gradient of J_p at $\mathbf{a}$ is

$$\nabla J_p(\mathbf{a}) = \sum_{j:\mathbf{a}^T\mathbf{y}_j \leq 0} -\mathbf{y}_j. \tag{6.6}$$

The update equation is

$$\mathbf{a}(t+1) = \mathbf{a}(t) + \eta \sum_{j:\mathbf{a}(t)^T\mathbf{y}_j \leq 0} \mathbf{y}_j,$$

where t is the iteration counter and the step length η is a parameter. This yields the Perceptron algorithm:

```
Perceptron algorithm
Set $t \leftarrow 0$, Initialize $\mathbf{a}(0), \eta, \epsilon$.
while $-\sum_{j:\mathbf{a}(t)^T\mathbf{y}_j \leq 0} \mathbf{a}^T\mathbf{y}_j > \epsilon$ do
    $\mathbf{a}(t+1) = \mathbf{a}(t) + \eta \sum_{j:\mathbf{a}(t)^T\mathbf{y}_j \leq 0} \mathbf{y}_j$.
    Set $t \leftarrow t + 1$
end while
Return $\mathbf{a}(t)$.
```

Two parameters - in addition to the initial value $\mathbf{a}(0)$ - need to be set: The stopping condition ϵ and the step length η. Sometimes the step length is called the learning rate. If training samples indeed are linearly separable, we can set ϵ to zero. The reason for this is that the Perceptron algorithm converges to the solution in a finite number of steps as long as the training samples are linearly separable. If the training samples are not linearly separable, the Perceptron algorithm does not converge and it is not a good choice in the linearly non-separable case. The step length can be set to 1.

Example. We illustrate the Perceptron algorithm by continuing the example of Section 6.3 and designing a linear classifier for those training samples. We select $\mathbf{a}(0) = [1, 1, 1]^T$. In the first iteration, samples $\mathbf{y}_4, \mathbf{y}_5$, and $\mathbf{y}_6$ are incorrectly classified. Hence,

$$\mathbf{a}(1) = \mathbf{a}(0) + \mathbf{y}_4 + \mathbf{y}_5 + \mathbf{y}_6 = [-2, -3, 6]^T.$$

By $\mathbf{a}(1)$, the sample number 3 is misclassified. Hence,

$$\mathbf{a}(2) = \mathbf{a}(1) + \mathbf{y}_3 = [-1, -1, 7]^T.$$

This is also the final result since all the training samples are now correctly classified. The resulting decision rule is: Classify $\mathbf{x}$ to ω_1 if $g(\mathbf{x}) = [-1, 7]\mathbf{x} - 1 > 0$ to ω_2 otherwise. It is staight-forward to determine the decision surface of the classifier based on the augmented weight vector. The decision surface is a line, because $d = 2$ and the classifier is linear. For the feature vectors $\mathbf{x}$ on the decision surface it holds that $g(\mathbf{x}) = 0$. The easiest way to draw the decision surface is detect two points from the surface (two points on a line define uniquely that line). For example, the feature vectors $[-1, 0]^T$ and $[6, 1]^T$ lay on the decision surface. Thus we can draw the decision surface based on these (see 6.3 in the section 6.6).

6.5 Perceptron for Multi-Class Case

We have concentrated only on the design of 2-class linear classifiers up to this point. Now, we would like to design a linear classifier for a c-class problem. We have augmented training samples from c classes:

$$\mathcal{D}_i = \{\mathbf{y}_{i1}, \ldots, \mathbf{y}_{in_i}\}, i = 1, \ldots, c.$$

We say that these training sets are linearly separable if there exists a linear classifier which classifies all the training samples correctly. The task is to find such (augmented) weight vectors $\mathbf{a}_1, \ldots, \mathbf{a}_c$ that always when $\mathbf{y} \in \mathcal{D}_i$, it holds that

$$\mathbf{a}_i^T \mathbf{y} > \mathbf{a}_j^T \mathbf{y} \tag{6.7}$$

for all $j \neq i$. In other words

$$\mathbf{a}_i^T \mathbf{y}_{ik} > \mathbf{a}_j^T \mathbf{y}_{ik},$$

for all $i = 1, \ldots, c, j \neq i$ and $k = 1, \ldots, n_i$. These inequalities can be manipulated to a 2-class and $c \cdot d$ feature problem by the *Kesler's construction*. This is a very useful theoretical tool but its use leads to computationally ineffective algorithms.

The Perceptron algorithm for the c-class case is:

```
c-class Perceptron algorithm
Set $t \leftarrow 0$, initialize $\mathbf{a}_1(0), \ldots, \mathbf{a}_c(0)$.
while Misclassified training samples exist do
  for all misclassified $\mathbf{y}_{ik}$ do
    $\mathbf{a}_i(t+1) = \mathbf{a}_i(t) + \mathbf{y}_{ik}$
    $\mathbf{a}_j(t+1) = \mathbf{a}_j(t) - \mathbf{y}_{ik}$   if   $\mathbf{a}_i(t)^T\mathbf{y}_{ik} \leq \mathbf{a}_j(t)^T\mathbf{y}_{ik}$
    $\mathbf{a}_l(t+1) = \mathbf{a}_l(t)$   if   $\mathbf{a}_i(t)^T\mathbf{y}_{ik} > \mathbf{a}_l(t)^T\mathbf{y}_{ik}$
    Set $t \leftarrow t+1$
  end for
end while
Return $\mathbf{a}_1(t), \ldots, \mathbf{a}_c(t)$.
```

As in the 2-class case, this algorithm works only if the training samples are linearly separable.

6.6 Minimum Squared Error Criterion

We still need an algorithm for designing linear classifiers when the training samples are not linearly separable. We will base the algorithm on the minimum squared error criterion and will now briefly introduce the criterion in the 2-class case.

We modify the training samples again as in Section 6.3. As a result we have a single training set of augmented feature vectors. We aim to minimize the criterion:

$$J_s(\mathbf{a}) = \sum_{i=1}^{n_1+n_2} (\mathbf{a}^T \mathbf{y}_i - 1)^2. \tag{6.8}$$

The criterion J_s can be written in the matrix-form as

$$J_s(\mathbf{a}) = ||Y\mathbf{a} - \mathbf{1}||^2, \tag{6.9}$$

where Y is the $n_1 + n_2 \times d + 1$ matrix consisting of the training samples, and $\mathbf{1}$ is $n_1 + n_2$ component vector consisting of only ones. Precisely

$$Y = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{n_1+n_2} \end{bmatrix}$$

and

$$\mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

This criterion has the unique minimum at

$$\mathbf{a} = (Y^T Y)^{-1} Y^T \mathbf{1}. \tag{6.10}$$

Note that we do not need to assume that Y would be non-singular or square. The selection of the vector $\mathbf{1}$ is based on the relation with the Bayes classifier. For more information on this, see section 5.8.3 in Duda, Hart and Stork.

Example Let's consider the training samples from the example in Section 6.3

and design a classifier based on these and Minimum Squared Error Criterion. Now

$$Y = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 1 \\ -1 & -1 & 1 \\ -1 & -1 & 2 \\ -1 & -2 & 2 \end{bmatrix}$$

and

$$\mathbf{1} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T.$$

We can compute the augmented weight vector using Matlab:

```
>> a = inv(Y'*Y)*Y'*ones(6,1)

a =

    0.0000
    0.0667
    0.6000
```

The discriminant function is $g(\mathbf{x}) = [0.0667, 0.6]\mathbf{x}$. Decision surface can be solved in the same manner as in the case of Perceptron criterion: points $[0,0]^T$ and $[-9,1]^T$ lay on the decision surface (see Fig. 6.3).

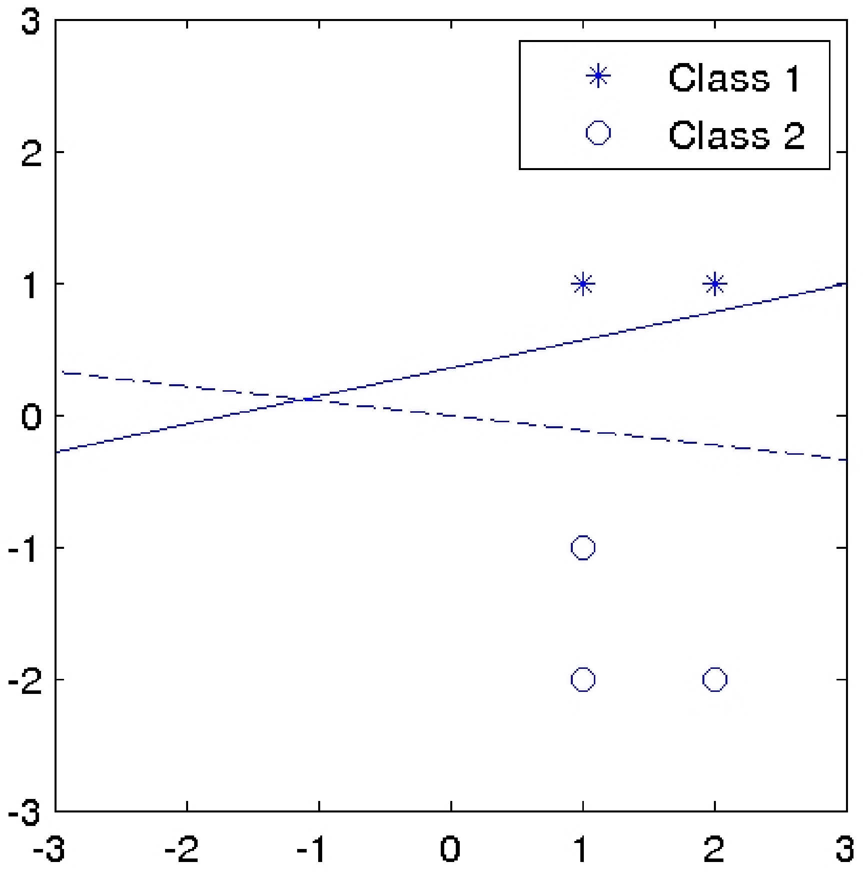

Figure 6.3: The decision surfaces based on the Perceptron criterion and and the Minimum Squared Error criterion. Training samples as in Section 6.3. The solid line is the decision surface based on the Perceptron citerion and the dashed line is the decision surface based on the Minimum Squared Error criterion.

Chapter 7

Classifier Evaluation

7.1 Estimation of the Probability of the Classification Error

It is important to estimate the error probability of the classifier when designing practical pattern recognition systems. This provides answers to important questions such as 'is the system good enough or does it need a further improvement?' and 'is the classifier A better than the classifier B?'.

Recall once more that we are interested in the probability that a test point which does not belong to the training set is mis-classified. This was termed *test error* in Section 5.5. Next, we introduce three ways to estimate the test error.

The *training error* is the frequency of error for the training data. The training error is a poor and overly optimistic estimate to the test error. For example, the training error of the NN-classifiers is automatically zero. This is obviously not true for the test error. The estimation of the test error by the training error is termed as the *resubstitution method*.

A much better estimate for the test error is obtained by dividing the training data

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_c$$

into two sets $\mathcal{D}_{training}$ and $\mathcal{D}_{test}$. We use $\mathcal{D}_{training}$ for the training of the classifier and $\mathcal{D}_{test}$ is solely used to estimate the test error. This method is called the *holdout method*. It is a good method when we have large number of training samples, and hence a modest decrease in the number of training samples does not essentially decrease the quality of the classifier. Note that the division into sets $\mathcal{D}_{training}$ and $\mathcal{D}_{test}$ must be randomized. For example, selecting $\mathcal{D}_{test} = \mathcal{D}_1$ is not a good idea.

In the case that we have only a few training samples, it is the best to use some cross-validation method to estimate the test error. The most simple of these is the *leave-one-out method*. In the leave-one-out method, we drop a single training sample from the training set and design a classifier based on the other training samples. Then, it is studied whether the dropped sample is classified correctly or not by the designed classifier. This is repeated for all n training samples. The test error estimate for the classifier α is the obvious

$$\hat{E}(\alpha) = \frac{v}{n},$$

where v is the number of misclassified training samples, and $n = \sum_{i=1}^{c} n_i$ is the total number of training samples. (The hat-notation refers to an estimate, i.e. $\hat{E}(\alpha)$ is an estimate of the true classification error $E(\alpha)$.) This method yields an accurate estimate for the test error. Its main disadvantage is the computational complexity: to estimate the test error for one classifier, we must design n classifiers.

7.2 Confusion Matrix

It is often worthwhile to study the *confusion matrix* in addition to the classification error. The element s_{ij} of the confusion matrix

$$S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1c} \\ \vdots & & \ddots & \vdots \\ s_{c1} & s_{c2} & \cdots & s_{cc} \end{bmatrix}$$

is the number of test samples that are assigned to the class ω_i but whose correct class would be ω_j. The order of indices i, j can be selected also differently, and it varies from one source reference to another. Hence, it pays off to be attentive. The confusion matrix can be estimated by the same methods as the test error.

Based on the confusion matrix, the classification (test) error can be obtained as

$$\hat{E}(\alpha) = \frac{\sum_{i=1}^{c} \sum_{j=1}^{c} s_{ij} - \sum_{i=1}^{c} s_{ii}}{\sum_{i=1}^{c} \sum_{j=1}^{c} s_{ij}}.$$

Based on the confusion matrix, also other important quantities for classifier design can be computed.

7.3 An Example

Let us illustrate the leave-one-out-method in the case of Fisher's Iris dataset. We study the classification based on ML-estimation, and we assume that the class conditional pdfs are Gaussian. The data was collected by separate sampling and therefore we assume that the prior probabilities for all classes are equal. This data can be loaded into Matlab by

```
>> load fisheriris
```

command. (At least, if the Statistics Toolbox is available). There should now be variables 'meas' and 'species' in Matlab. The variable 'meas' contains the feature vectors from 150 Irises and the variable 'species' contains the class of each feature vector. The feature vectors numbered from 1 to 50 belong to the class ω_1 (iris setosa), the feature vectors numbered from 51 to 100 belong to the class ω_2 (iris versicolor), and the feature vectors numbered from 101 to 150 belong to the class ω_3 (iris virginica). This simplifies the implementation of the leave-one-out method in Matlab. The following piece of Matlab-code estimates the confusion matrix by the leave-one-out method:

```
% initialize the confusion matrix
s = zeros(3,3);
% class 1, feature vectors 1 - 50
% class 2, feature vectors 51 - 100
% class 3, feature vectors 101 - 150
for i = 1:150
  trueclass = ceil(i/50);
  test = meas(i,:); % test point is feature vector number i
  % to the class 'trueclass' training data contains all feature
  % vectors from that class except the test point
  % for other classes all training data is used.
  train1 = [meas(1:min((i - 1),50),:);meas((i + 1):50,:)];
  train2 = [meas(51:min(i - 1,100),:);meas(max((i + 1),51):100,:)];
  train3 = [meas(101:(i - 1),:);meas(max(101,(i + 1)):150,:)];

  % Train the classifier, i.e. compute the means vectors and
  % covariance matrices for each class
  mu1 = mean(train1);
  cov1 = cov(train1);
  mu2 = mean(train2);
  cov2 = cov(train2);
  mu3 = mean(train3);
  cov3 = cov(train3);
  % the classifier has been trained. Now compute the posterior
% probabilities and see to what class the test point is classified
% to
  posterior(1) = mvnpdf(test,mu1,cov1)*(1/3);
  posterior(2) = mvnpdf(test,mu2,cov2)*(1/3);
  posterior(3) = mvnpdf(test,mu3,cov3)*(1/3);
  [maxposterior,testclass] = max(posterior);
  % and increment the corresponding element of the confusion matrix by
  % one.
  s(testclass,trueclass) = s(testclass,trueclass) + 1;
end
```

Above, 'ceil(x)' returns the least integer that is larger than the real number x. We have used also some Matlab specific programming tricks: for example `meas(56:50,:)` is an empty matrix. About 'mvnpdf', Matlab's help states the following

```
Y = MVNPDF(X,MU,SIGMA) returns the density of the multivariate normal
    distribution with mean MU and covariance SIGMA, evaluated at each row
    of X.
```

The confusion matrix is

```
s =

    50     0     0
     0    47     1
     0     3    49
```

And the estimate for the test error is accordingly:

$$\hat{E}(\alpha) = 4/150 \approx 0.027.$$

Chapter 8

Unsupervised Learning and Clustering

8.1 Introduction

Until now, we have assumed that we have a set of correctly labeled training samples for training the classifier. These procedures using training samples are part of *supervised classification* (see Section 2.4). However, in some circumstances it is necessary to resort to *unsupervised classification* that is also termed as *clustering.* These procedures do not use labeled training data. Instead, we have a collection of unlabeled samples, and we try to classify them based only on the features in data. In other words: there is no explicit teacher. Clearly, clustering is a more difficult problem than supervised classification. However, there are certain situations where clustering is either useful or necessary. These include:

1. The collection and classification of training data can be costly and time consuming. Therefore, it can be impossible to collect a training set. Also, when there are very many training samples, it can be that all of these cannot be hand-labeled. Then, it is useful to train a supervised classifier with a small portion of training data and use then clustering procedures to fine tune the classifier based on the large, unclassified dataset.

2. For data mining, it can be useful to search for natural clusters/groupings among the data, and then recognize the clusters.

3. The properties of feature vectors can change over time. Then, supervised classification is not reasonable, because sooner or later the test feature vectors would have completely different properties than the training data had. For example, this problem is commonplace in the processing of medical images.

4. The clustering can be useful when searching for good parametric families for the class conditional densities for the supervised classification.

Above situations only are examples of the situations where clustering is worthwhile. There are many more situations requiring clustering.

8.2 The Clustering Problem

We define the clustering problem more precisely. We have a set of feature vectors

$$\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}.$$

The task is to place each of these feature vectors into one of the c classes. In other words, the task is to find c sets $\mathcal{D}_1, \ldots, \mathcal{D}_c$ so that

$$\bigcup_{i=1}^{c} \mathcal{D}_i = \mathcal{D}$$

and

$$\mathcal{D}_j \cap \mathcal{D}_i = \emptyset$$

for all $i \neq j$.

Naturally, we need some further assumptions for the problem to be sensible. (An arbitrary division of $\mathcal{D}$ into different classes is not likely to be useful.) Here, we assume that we can measure the similarity of any two feature vectors somehow. Then, the task is to maximize the similarity of feature vectors within a class. For this, we need to be able to define how similar are the feature vectors belonging to some set $\mathcal{D}_i$.

In the following sections, we will introduce two alternatives for defining how similar a certain set of feature vectors is. We have already seen their counterparts in supervised classification. The first one, the *k-means clustering*, reminds minimum distance classification. The second one, the *expectation maximization algorithm*, is rooted on ML-estimates and *finite mixture models*

8.3 K-means Clustering

8.3.1 K-means Criterion

Recall that the minimum distance classifier assigned a test point to the class with the nearest mean to the test point. Denote now the number of feature vectors in the class $\mathcal{D}_i$ by n_i. (We do not know the value of n_i yet.).

The problem is that we do not know the mean vectors. Despite of this, define the similarity of $\mathcal{D}_i$ as

$$s(\mathcal{D}_i) = -\sum_{\mathbf{x} \in \mathcal{D}_i} ||\mathbf{x} - \mu_i||^2,$$

where $\mu_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{D}_i} \mathbf{x}$. We solve the unknown mean vectors by maximizing the sum of similarities of all classes. Equivalently, we can minimize the negative of this sum:

$$J(\mathcal{D}_1, \ldots, \mathcal{D}_c) = \sum_{i=1}^{c} \sum_{\mathbf{x} \in \mathcal{D}_i} ||\mathbf{x} - \mu_i||^2. \tag{8.1}$$

This is *the k-means criterion.*

8.3.2 K-means Algorithm

Note that based on the knowledge of clusters $\mathcal{D}_1, \ldots, \mathcal{D}_c$, we can compute the mean vectors. Conversely, based on the knowledge of mean vectors, we can compute the clusters. This observation is important when minimizing the criterion (8.1). The criterion J is a function of the partition of the discrete set $\mathcal{D}$, and therefore the steepest descent method of Chapter 6 cannot be used. The algorithm to minimize the criterion (8.1) is termed the *k-means algorithm.* The basic principle is to compute the mean vectors based on the current clustering, re-classify the feature vectors based on updated mean vectors, and repeat these two steps until the clustering does not change anymore.

```
k-means algorithm
Initialize $\mu_1(0), \ldots, \mu_c(0)$, set $t \leftarrow 0$
repeat
  Classify each $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to the class $\mathcal{D}_j(t)$ whose mean vector $\mu_j(t)$ is the nearest
  to $\mathbf{x}_i$.
  for $k = 1$ to $c$ do
    update the mean vectors $\mu_k(t+1) = \frac{1}{|\mathcal{D}_k(t)|} \sum_{\mathbf{x} \in \mathcal{D}_k(t)} \mathbf{x}$
  end for
  Set $t \leftarrow t + 1$
until clustering did not change
Return $\mathcal{D}_1(t-1), \ldots, \mathcal{D}_c(t-1)$.
```

Notes: 1) $\mathcal{D}_j(t-1)$ is returned because the iteration counter was incremented by one just before the termination of the algorithm . 2) The notation $|\mathcal{D}_k(t)|$ refers to the number of elements in the set $\mathcal{D}_k(t)$.

Typically, the mean vectors are initialized as randomly selected feature vectors. For example, we can set $\mu_j(0) \leftarrow \mathbf{x}_j$. Also, the result of the algorithm depends on the initialization. This is because the k-means algorithm is a local optimization algorithm and the minimization problem in this case has several local minima. (See Figure 8.1). In the case of the k-means criterion, these local minima are not too numerous.

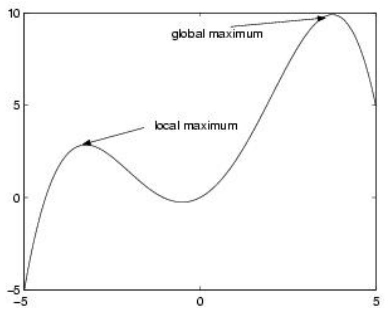

Figure 8.1: Local vs. global maximum

8.3.3 Properties of the K-means Clustering

The K-means clustering is 'an application' of the minimum distance classifier to clustering. Therefore, the k-means clustering suffers from the same drawbacks as the minimum distance classification. If the clusters are of different sizes (i.e. they contain a different number of feature vectors), the k-means clustering is not successful. If the clusters have very different covariances or scales of different features are different, then the clustering by k-means is not appropriate. See figure 8.2. Note that these problems are due to the criterion function and not the algorithm minimizing it. Because of these problems, we need a more complicated clustering criterion/algorithm to solve clustering problems such as those depicted in Figure 8.2. Instead, k-means performs well when clusters form compact clouds that are well separated from one another.

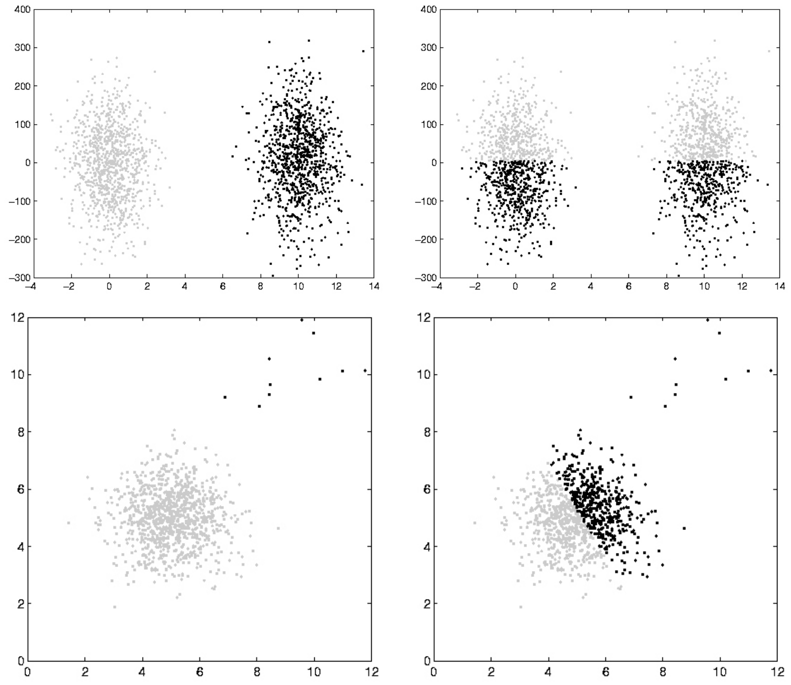

Figure 8.2: Examples of problematic clustering tasks for k-means. Top row: The scale of features is different (note the scaling of the axis). Bottom row: There are many more feature vectors in one class than in the other. 'True classifications' are shown on left and k-means results are on right.

8.4 Finite Mixture Models

We consider Gaussian finite mixture models (FMMs)[1] for solving clustering problems that are out of the scope of K-means. The task is now to learn the Bayes classifier

[1]Finite mixture models can be derived also for other families of parametric pdfs, but we will just consider FMMs based on normal densities during this course.

in the unsupervised manner (without any training samples) when the parametric families of the class conditional densities are known. We will assume that:

1. The class conditional densities are modeled by normal densities, i.e. $p(\mathbf{x}|\omega_j) = p_{normal}(\mathbf{x}|\mu_J, \Sigma_j)$ for all $j = 1, \ldots, c$.
2. Parameters $\mu_1, \ldots, \mu_c, \Sigma_1, \ldots, \Sigma_c$ are not known.
3. Priors $P(\omega_1), \ldots, P(\omega_c)$ are not known.
4. There are no training samples: i.e. the classes of feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are not known, and the task is to classify the feature vectors.

We assume that feature vectors are occurrences of independent random variables. Each random variable is distributed according to some class conditional density but we do not know according to which one. We can, however, write the probability density that $\mathbf{x}_i$ is observed and it belongs to some class. Firstly,

$$p(\mathbf{x}_i, \omega_j) = P(\omega_j)p(\mathbf{x}_i|\omega_j)$$

for all i and j. Secondly

$$p(\mathbf{x}_i) = \sum_{j=1}^{c} p(\mathbf{x}_i, \omega_j),$$

because $\mathbf{x}_i$ has to belong to exactly one of the classes $\omega_1, \ldots, \omega_c$. (The left hand side of the above equation is a marginal density.) By combining these, we obtain

$$p(\mathbf{x}_i) = \sum_{j=1}^{c} p(\mathbf{x}_i|\omega_j)P(\omega_j). \tag{8.2}$$

This is a *mixture density.* The priors $P(\omega_i)$ in mixture densities are called *mixing parameters*, and the class conditional densities are called *component densities.*

We assumed that the component densities are normal densities. Hence, we obtain the parametric mixture density

$$p(\mathbf{x}_i|\theta) = \sum_{j=1}^{c} p_{normal}(\mathbf{x}_i|\mu_j, \Sigma_j)P(\omega_j), \tag{8.3}$$

where the parameter vector

$$\theta = (\mu_1, \ldots, \mu_c, \Sigma_1, \ldots, \Sigma_c, P(\omega_1), \ldots, P(\omega_c))^T.$$

We can then derive an algorithm to solve the parameter vector $\hat{\theta}$ based on ML estimation. We will introduce the algorithm in the next section. Right now, we go a bit further and study how $\mathbf{x}_i$ can be classified when we have ML estimates for all the parameters. We will use the Bayes classifier as in the section 5.2, that is we substitute the unknown parameters by their estimated values. This leads to the decision rule

$$\alpha_{mixture}(\mathbf{x}_i) = \arg \max_{\omega_j, j=1,\ldots,c} p(\mathbf{x}_i|\hat{\mu}_j, \hat{\Sigma}_j)\hat{P}(\omega_j).$$

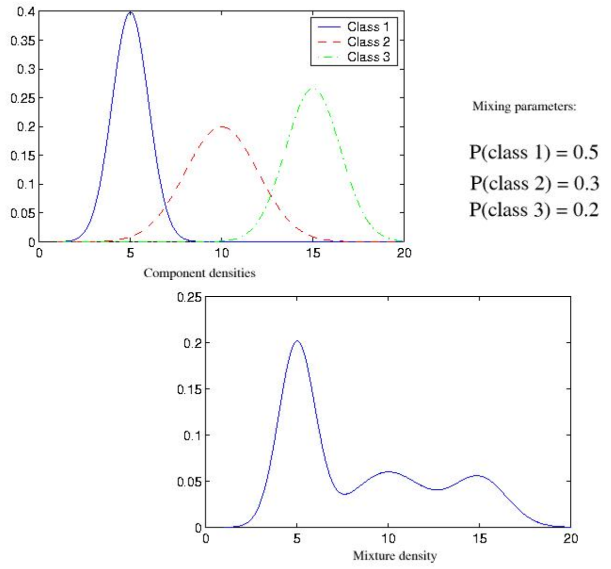

Figure 8.3: Finite mixture models. On top component densities are shown. On bottom the resulting mixture density when component densities and mixing parameters are as shown in the top panel.

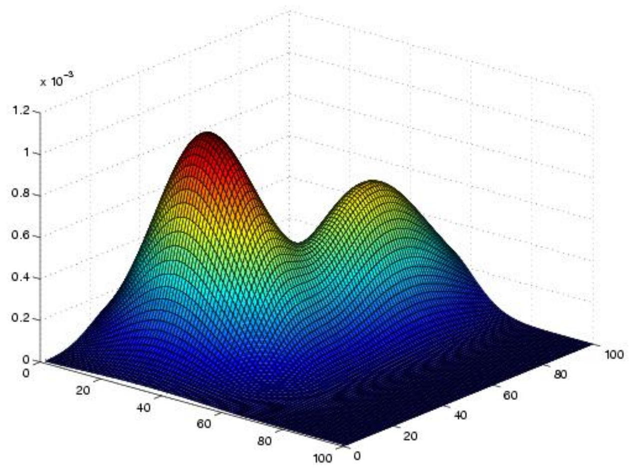

Figure 8.4: A finite mixture model in the case of two features.

A problem with FMMs is that they are not always *identifiable*. This means that two different parameter vectors θ_1 and θ_2 may result in exactly the same mixture density, i.e.

$$p(\mathbf{x}|\theta_1) = p(\mathbf{x}|\theta_2)$$

for all $\mathbf{x}$. However, the classifier derived based on the parameter vector θ_1 can still be different than the classifier that is based on θ_2. And therefore, this is a serious problem because the parameter vectors cannot be distuinguished from each other based on ML estimation.. *Switching problems* are special cases of identifiability problems. If we switch the classes ω_i and ω_j, we end up with exactly the same mixture density as long as component densities for ω_i and ω_j belong to the same parametric family. Therefore, the classes must be named (or given interpretation) after the classification. This is problematic, if we are after a completely unsupervised classifier.

8.5 EM Algorithm

We return to solving the ML estimation problem. Because $\mathbf{x}_1, \ldots, \mathbf{x}_n$ were occurrences of independent random variables, we can write the likelihood function as

$$\prod_{i=1}^{n} p(\mathbf{x}_i|\theta) = \prod_{i=1}^{n} \sum_{j=1}^{c} p_{normal}(\mathbf{x}_i|\mu_j, \Sigma_j)P(\omega_j).$$

Maximizing this with respect to the parameter vector θ results in the ML estimate $\hat{\theta}$. Once again, it is more convenient to deal with the log-likelihood function, i.e.

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \ln[\sum_{j=1}^{c} p_{normal}(\mathbf{x}_i|\mu_j, \Sigma_j)P(\omega_j)]. \tag{8.4}$$

This is a challenging maximization task. (Actually, we are interested in the largest local maximum, because the (log)-likelihood function is not bounded from above, see e.g. Duda, Hart, Stork, Chapter 10). However, we can derive a expectation-maximization (EM) algorithm for solving (8.4). We will skip the derivation. The algorithm can be given as:

EM algorithm

1. Initialize $\mu_j^0, \Sigma_j^0, P^0(\omega_j)$, set $t \leftarrow 0$.
2. (E-step) Compute the posterior probabilities that $\mathbf{x}_i$ belongs to the class ω_j

$$p_{ij}^{t+1} = \frac{P^t(\omega_j) p_{normal}(\mathbf{x}_i | \mu_j^t, \Sigma_j^t)}{\sum_{k=0}^{c} P^t(\omega_k) p_{normal}(\mathbf{x}_i | \mu_k^t, \Sigma_k^t)}.$$

3. (M-step) Update the parameter values

$$P^{t+1}(\omega_j) = (1/n) \sum_i p_{ij}^{t+1} \tag{8.5}$$

$$\mu_j^{t+1} = \frac{\sum_i p_{ij}^{t+1} \mathbf{x}_i}{\sum_i p_{ij}^{t+1}} \tag{8.6}$$

$$\Sigma_j^{t+1} = \frac{\sum_i p_{ij}^{t+1} (\mathbf{x}_i - \mu_j^{t+1})(\mathbf{x}_i - \mu_j^{t+1})^T}{\sum_i p_{ij}^{t+1}} \tag{8.7}$$

4. Set $t \leftarrow t + 1$.
5. Stop if some convergence criterion is fulfilled, otherwise return to step 2.

In most cases, The EM algorithm finds a local maximum of the likelihood function. However, there are usually several local maxima and hence the final result is strongly dependent on the initialization.