Dipartimento di Informatica e
Scienze dell'Informazione

A Trainable System for Event Detection and Recognition

by

Massimiliano Pittore

Theses Series DISI-TH-2000-

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

A Trainable System for Event Detection and Recognition

by

Massimiliano Pittore

January, 2000

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35
I-16146 Genova, Italy
`http://www.disi.unige.it/`

Ph.D. Thesis in Computer Science

Submitted by Massimiliano Pittore
DISI, Univ. di Genova
`pittore@disi.unige.it`

Date of submission: December 1999

Title: A Trainable System for Event Detection and Recognition

Advisor:
Leila De Floriani
`deflo@disi.unige.it`

Supervisor: Alessandro Verri
DISI
`verri@disi.unige.it`

Ext. Reviewers: Manuel Trucco
Heriot-Watt University, Edimburgh (UK)
`mtc@cee.hw.ac.uk`

Patrick Bouthemy
INRIA-IRISA, Rennes (F)
`patrick.Bouthemy@inria.fr`

Table of Contents

List of Figures

Abstract

Statistical learning theory has emerged in the last few years as a solid and elegant framework for addressing the problem of learning from examples. This theory characterizes the necessary and sufficient conditions for a learning algorithm to be consistent. The key concept is the *capacity* of the set of hypotheses employed in the learning algorithm and the goal is to control the capacity depending on the number of examples. Structural Risk Minimization (SRM) is the main theoretical algorithm which implements this idea. For practical purposes, however, SRM is a very hard problem and impossible to implement when dealing with a large number of examples. Recently Support Vector Machines (SVM) have been proposed as a practical algorithm implementing the idea of capacity control. Although SVM are only an approximation of SRM, the experimental results in many different applications confirm – specially for pattern recognition – the power of this technique.

In this thesis we introduce a trainable system we have developed in order to exploit the flexibility of SVMs in practical applications. The system detects dynamic events through a low-level vision module, and uses SVMs for both regression and classification in order to redpresent and recognize the detected events. For testing the system we chose two problems in which the key properties of the learning machine are the *generalization capability* (for efficiently dealing with small *training sets*) and the effectiveness of the event description. The developed system has been successfully applied to these problems showing the goodness of the choice of SVMs as learning machines in practical applications.

Chapter 1

Introduction

We introduce the problem of learning from examples, with particular emphasis on the Computer Vision applications based on dynamic events. Then we discuss the framework in which this problem is studied, and provide an outline of the thesis.

1.1 Aim

A key property of natural visual systems is the ability to perform the same tasks in different environments. On the contrary, one of the main drawbacks of the computer vision systems developed over the last decades is the lack of flexibility with respect to changes of the observed environment. This is hardly surprising if the formidable difference between the complexity of natural and artificial visual systems is taken into due account. A promising way to bridge this gap is the development of systems that can be *trained* instead of *programmed* to perform the required tasks.

This thesis we introduce a system, which we call *VIDERE* (from *VIsual Dynamic Event REcognition*), able to learn to represent and recognize visual dynamic events from examples. *VIDERE* detects dynamic events through a simple low-level vision module and then represent and recognize a detected event by means of Support Vector Machines (SVMs). SVMs, a new method for solving function estimation problems recently introduced by Vapnik (1995) (see also Cortes and Vapnik, 1995), are attracting increasing attention in both the machine learning (Blanz et al. 1996, Scholkopf et al. 1996) and computer vision community (Osuna, Freund and Girosi 1997, Pontil and Verri 1998) for their theoretical and practical properties.

The aim of this work is *(a)* to show that, notwithstanding the structural complexity of visual tasks, a trainable computer vision system can reach very good level of performances from a reasonably small number of examples, and *(b)* to assess the potential of SVMs as a tool for learning solution of complex computer vision problems.

1.2 Statistical learning theory

This thesis address the problem of pattern recognition from a statistical viewpoint. According to statistical learning theory, the *learning* process is equivalent to the problem of recovering a mapping between two random variables, $\mathbf{x} \in X \subseteq \Re^n$ (the *pattern*)) and $y \in Y \subseteq \Re$ (y is the outcome of the *recognition* of pattern $\mathbf{x}$), that are related by a probabilistic relationship. This relationship is probabilistic because generally an element of X does not determine uniquely an element of Y, but rather a probability distribution on Y. This can be formalized assuming that a probability distribution $P(\mathbf{x}, y)$ is defined over the set $X \times Y$. In practice, however, the probability distribution $P(\mathbf{x}, y)$ is *unknown* and we are only provided with a finite set of examples of this probabilistic relationship, an *experimental* data set

$$T \equiv \{(\mathbf{x}_i, y_i) \in X \times Y,\ i = 1, \ldots, l\},$$

called the *training set*, obtained by sampling l times the set $X \times Y$ according to $P(\mathbf{x}, y)$. The problem of learning consists in, given the data set T, providing an *estimator*, that is a function $f : X \to Y$, that can be used, given any value of $\mathbf{x} \in X$, to predict a value $y \in Y$. In the case $Y = \{-1, 1\}$ we have the binary pattern classification (or classification) problem. When Y is continuous, the problem is known as *regression*.

In statistical learning theory (see Vapnik, 1998 for a full review), the standard way to solve the learning problem consists in defining a *risk functional*, which measures the average amount of error associated with an estimator, and then to look for the estimator, among the allowed ones, with the lowest risk. If $L(y, f(\mathbf{x}))$ is the loss function measuring the error we make when we predict y by $f(\mathbf{x})$, then the average error is the so called *expected risk*:

$$R_f \equiv \int L(y, f(\mathbf{x}))P(\mathbf{x}, y)\ d\mathbf{x}dy \tag{1.1}$$

The function $f(\mathbf{x}) = \arg\min_{\mathcal{F}} R_f$ which minimizes the expected risk is selected from a *function space* $\mathcal{F}$.
In the case of classification $\mathcal{F}$ is a set of functions from X to $\{-1, 1\}$ which are called indicator functions and the loss function is simply $V(y, f(\mathbf{x})) = \theta(yf(\mathbf{x}))$.

Unfortunately this function cannot be found in practice, because the probability distribution $P(\mathbf{x}, y)$ that defines the expected risk is unknown, and only a sample of it, the data set T, is available. Therefore we need an *induction principle* that we can use to *infer* the unknown function from the limited number of training data we have. To overcome this limitation statistical learning theory intriduces the so-called *empirical risk minimization (ERM)* induction principle. The ERM method consists in using the data set T to build a stochastic approximation to the expected risk, which is usually called the *empirical risk*, and is defined as:

$$R_{\mathrm{emp}} = \frac{1}{l} \sum_{i=1}^{l} L(y_i, f(\mathbf{x}_i)). \tag{1.2}$$

We denote by $\hat{f}$ the minimizer of the empirical risk in $\mathcal{F}$. An important problem of the theory is whether the minimum of the empirical risk is close in probability to the expected risk of $\hat{f}_l$, that is whether the inequality:

$$R(\hat{f}) < R_{emp}(\hat{f}) + \epsilon \tag{1.3}$$

holds with high probability η and for a small value of ϵ. If this is the case, then we can use the the function $\hat{f}$ to predict the value y of a new input vector $\mathbf{x}$ with high confidence. Intuitively, the more "complex" are the functions of the considered space $\mathcal{F}$, the better they will fit the experimental data that form the *training* set. At the same time, spaces of highly complex functions, or spaces with high *capacity* will not achieve a good result in minimizing the *actual* risk because, in rough words, they spend their efforts in keeping low the *empirical* risk on a limited sample of an unknown distribution. In order to find out the best trade off between high *capacity* and low *actual risk*, a formal *capacity* measure has to be introduced. In the case of classification the concept of capacity is simple: The capacity or VC-dimension (Vapnik and Chervonenkis, 1971) of a set of indicator functions is the maximum number h of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_h$ that can be separated into two classes in all 2^h possible ways using functions of the set. If, for any number N, it is possible to find N points that can be separated in all the 2^N possible ways, we will say that the VC-dimension of the set is infinite.

The concept of capacity can be extended in several ways to the case of regression, that is to the case where $\mathcal{F}$ is a set of real value functions.

1.3 A new inductive principle: minimizing the *Structural Risk*

It is possible to derive equations that relate ϵ to the capacity h, the number of examples l and the probability η. From a qualitative point of view, all these results basically say that, for a fixed value of η, ϵ is an increasing function of h and a decreasing function of l. Moreover, for all practical purposes ϵ can be treated as a function of $\frac{h}{l}$ only.

Vapnik and Chervonchensis (1971) have shown that for the classification problem the following bound holds with high probability:

$$R(\hat{f}) < R_{emp}(\hat{f}) + \sqrt{\frac{h}{l} \ln \frac{l}{h}}. \tag{1.4}$$

If the number of samples l tends to infinite, the *empirical risk* leads the bound 1.4. But in practice often the condition $\frac{h}{l} < 20$ (small samples) holds, and the right hand side of 1.4 has to be minimized over both terms. This observation leads to the method of *Structural Risk Minimization (SRM)*: not only minimize the empirical risk, but instead minimize a combination of the empirical risk and the *complexity of the hypothesis space* measured by the second term in the r.h.s. of inequality (1.4).

The idea of SRM is to define a nested sequence of function spaces $S_1 \subset S_2 \subset \ldots \subset S_{n(l)}$ with $n(l)$ a non-decreasing integer function of l where each hypothesis space S_i has VC-dimension finite and larger than that of all previous sets, i.e. if h_i is the VC-dimension of space S_i, then $h_1 \leq h_2 \leq \ldots \leq h_{n(l)}$. For example S_i could be the set of polynomials of degree i, or a set of splines with i nodes, or some more complicated nonlinear parameterization. For each element S_i of the structure the solution of the learning problem is:

$$\hat{f}_{i,l} = \arg \min_{f \in S_i} R_{\text{emp}}(f). \tag{1.5}$$

Because of the way we defined our structure it is clear that the larger i is the smaller the empirical error of $\hat{f}_{i,l}$ is (since we have greater "flexibility" to fit our training data), but the larger the VC-dimension part (second term) of the r.h.s. of (1.4) is. Using such a nested sequence of more and more complex function spaces, the SRM learning technique consists of choosing the space $S_{n^*(l)}$ for which the right hand side of inequality (1.4) is minimized.

The method of SRM can be applied in several ways to the regression case. It only requires to extend the concept of capacity for real value functions. We will briefly discuss this extension in chapter 3.

1.4 A "special" subset of training set: *Support Vectors*

Unfortunately, the implementation of SRM method is computationally expensive for many practical applications and impossible in the case where the training set is large, say $l > 10^4$.

Various algorithms which approximatively implement the idea of SRM have been proposed. An approach that is loosely inspired to SRM principle has been proposed by Vapnik (Cortes and Vapnik 1995). It looks for a minimum of the following functional:

$$H[f] = \frac{1}{l}\sum_{i=1}^{l} V(y_i, f(\mathbf{x}_i)), \tag{1.6}$$

The problem is formulated as a quadratic programming problem, and is solved in the context of constrained optimization, as we see in chapter 3. In the Vapnik approach the minimizer of functional (1.6) has, independently on the loss function V, the same general form

$$f(\mathbf{x}) = \sum_{i=1}^{l} c_i K(\mathbf{x}, \mathbf{x}_i), \tag{1.7}$$

where we have defined the symmetric kernel K an be viewed as the inner product of some Hilbert space (for being an *admissible* kernel it has also to satisfy the so called *Mercer* condition, that we will see in chapter 3). Notice that equation (1.7) establishes a representation of the function f as a linear combination of kernels centered in each data point.

The coefficients c_i in equation (1.7) are generally the solution of a system of nonlinear equations.

The resulting learning machine is called *Support Vector Machines*, a learning technique which has recently been proposed for both classification (Cortes and Vapnik, 1995) and regression (Vapnik, 1995; Vapnik, Golowich, and Smola, 1996) depending on the Loss function:

- Support Vector Machines Regression (SVMR)

$$V(y_i, f(\mathbf{x}_i)) = |y_i - f(\mathbf{x}_i)|_\epsilon^\sigma, \tag{1.8}$$

- Support Vector Machines Classification (SVMC)

$$V(y_i, f(\mathbf{x}_i)) = \theta(1 - y_i f(\mathbf{x}_i))(1 - y_i f(\mathbf{x}_i))^\sigma, \tag{1.9}$$

where $\theta(\cdot)$ is the Heaviside function and the function $|\cdot|_\epsilon$, called epsilon-insensitive loss, is defined as follows:

$$V(x) \equiv |x|_\epsilon \equiv \begin{cases} 0 & \text{if } |x| < \epsilon \\ |x| - \epsilon & \text{otherwise.} \end{cases} \tag{1.10}$$

The parameter σ is typically set equal to 1 or 2. In this case it can be shown that the coefficients c_i are the solution of a quadratic programming problem with box constraints. In particular the size of the box is inversely proportional to the regularization parameter λ.

A remarkable feature of the SVM technique is that the loss functions (1.8) and (1.9) determine quite different properties of the solution which is, unlike in the case of standard regularization, sparse in the c_n (Girosi 1998). With this we mean that only a (typically) small fraction of the coefficients c_i are nonzero. In order to gain an intuition about this phenomenon let us consider the regression case: note that the epsilon-insensitive loss assigns zero cost to errors smaller then ϵ. In other words, for the cost function $|\cdot|_\epsilon$ any function closer than ϵ to the data points is a perfect interpolant. We can think to the parameter ϵ as the resolution at which we want to look the data. For this reason we expect that the larger ϵ is, the more sparse the representation will be.

The data point $\mathbf{x}_i$ associated with the nonzero coefficients in equation (1.7) are called *support vectors*. The interpretation of the support vectors is more intuitive in the case of classification. In this case the function in equation (1.7) can be seen as a boundary surface in the input space X in the sense that points such that $f(\mathbf{x}) > 0$ (resp. $f(\mathbf{x}) < 0$) are classified in the class 1 (resp. -1). The support vectors are those data points that lie near the boundary surface. More precisely let us assume for the moment that the training set is separable, that is $f(\mathbf{x}_i) = y_i, \ \forall \ i = 1, \ldots, l$ Then it can be shown that the support vectors are the nearest points to the boundary surface, where the distance is given by:

$$d = \frac{1}{\Omega(f)},$$

with $\Omega(f)$ the norm of f in the space where is defined. If the training set is not separable the support vectors are either the data points at a distance less that d from the boundary or the points which are misclassified. In both cases if all the data points which are not support vectors were discarded from the training set the minimum of functional (1.6) and the solution (1.7) would be unchanged.

In conclusion, the support vectors condense all the information contained in the training set S which is needed to classify new data points. An interesting perspective on SVM is to consider its information compression properties. The support vectors represent in this sense the most informative data points and compress the information contained in the training set: for the purpose of, say, classification of future vectors only the support vectors need to be stored, while all other training examples can be discarded[1].

If the training set is not separable the support vectors are either the data points at a distance less that d from the boundary or the points which are misclassified. In both cases if all the data points which are not support vectors were discarded from the training set the minimum of functional described before and the solution would be unchanged. In this context, This is a key property of SVM and might explain why this technique works well in many practical applications. A similar analysis holds for the regression case but a clear geometric interpretation is still missing.

1.5 Application of *SVM*'s to *Dynamic Event Detection*

In this thesis we address the problem of *detecting* and *classifying* dynamic events from a sequence of images. The developed system is heavily based on Support Vectors Machine. SVM's are used both for providing a suitable *event description* from the raw data, and for classifying the detected events according to the supervisor's rule. We applied the *VIDERE* system to two different Computer Vision tasks:

1. detecting and recognizing people motion in an indoor environment
2. recognizing simple arm gestures

For each of these applications the meaning, the structure and the chosed description of the dynamic events is different, but the system is flexible enough to be used in both cases without any modification (except for the very low-level image processing modules). Let us briefly introduce the considered applications.

1.5.1 Detecting and recognizing people motion

In this case, the images are acquired by a camera looking down a corridor from a fixed position. Occasionally, one or more people are expected to walk up or down the corridor, possibly entering or leaving one of the adjacent rooms. The goal is to build a system able to learn from examples to

[1]There is in fact a relation between the compression factor expressed as the ratio of data points to support vectors and the probability expected risk of the SVM (see Vapnik, 1998). More in general, Vapnik, in comparing the empirical risk minimization principle with the Minimum Description Length principle (Rissanen, 1978), derives a bound on the generalization error as a function of the compression coefficient.

- detect the presence of moving people in the observed scene, and
- recognize their trajectory.

The ability of learning to perform the given tasks from examples endows the system with the flexibility sufficient to effectively work in different environments and perform different tasks.

Let us give an operative definition of dynamic event and of the generic tasks to be performed. During the training stage a number of image sequences is acquired. In each sequence a person walks in the corridor and performs one of n possible *actions* labeled as action $1, 2, ...,$ or n. These actions are *dynamic events* and, in plain words, could be described as "somebody is crossing the corridor going from room A to B", or "somebody is walking up the corridor and entering room C". In the test stage the system acquires a new image sequence and is expected to properly detect and classify each of the observed actions, if any, as one of the n possible actions learned in the training stage.

In order to perform these tasks, in both the training and test stages, the system should be able to

(1) detect the beginning and end of a dynamic event, and

(2) compute a stereotyped representation of the observed event.

While in the training stage the event label is known *a priori*, in the test stage the system should also be able to

(3) classify the detected event on the basis of the computed representation.

To this purpose we built a system consisting of three modules, a *detection*, a *representation*, and a *recognition* module, which will be described in more details in chapter 6. Both the *representation*, and the *recognition* modules have been implemented using *Support Vector* based Learning machines.The reasons for this choice are:

- the number of available samples is very small ($o(10^1 - 10^2)$) when compared with the size of a "usual" training set in other computer vision applications ($o(10^3 - 10^6)$)
- the Learning machines based on SV (called *Support Vector Machines* endow great flexibility in the choice of the functions to be used for classification, and allow a syntethic but powerful description of decision boundaries in terms of few elements of the training set.

1.5.2 Recognizing arm gestures

The scenario of the second application is very simple: someone is standing in front of the camera, moving only its left arm. Only three kinds of movements are allowed: moving the left arm to the left at three different heighs, denoted as “low”, “medium”, “hight”.
The difficult arises when different people are asked to move their arm following a verbal suggestion (such as “move the arm on the left, down”). In fact the representations of events from the three classes reflect the heterogeneity of the gesture interpretation.
Therefore this problem, despite its triviality, turns out to be an interesting benchmark of the system preformances.

1.6 Outline of the thesis

The thesis is organized as follows: In chapter 2 we present the pattern recognition problem in its Bayesian formulation. In Chapter 3,4 we overview the fundamental results of statistical learning theory, and, with particular emphasis on the induction principle as *Empirical and Structural Risk Minimization* In Chapter 5 we introduce and discuss the *Support Vector Machines* (SVM) for classification and for regression, as an application of statistical learning to the pattern recognition problem.

In Chapter 6 we discuss the main algorithms for implementing SVM for classification problems In Chapter 7 we introduce the system *Videre* and present two applications of SVMs to the problem of *Dynamic Event Description and Recognition.* A brief Conclusion will follow.

Chapter 2

Classical techniques for Pattern Recognition

2.1 The Problem

The problem we want to address is stated as follows. Given a set of measurements, obtained through observations and represented as a pattern vector $\mathbf{x}$, we wish to assign the pattern to one of C possible classes, $\omega_i, i = 1, \ldots, C$. If Ω is the observation space, we need a *rule* for correctly assigning a point in this space to a class. Such a *Partition Rule* will partition the space Ω into C regions $\Omega_i, i = 1, \ldots, C$. If an observation (a vector $\mathbf{x}$) is in Ω_i, then it is assumed to belong to the class ω_i. Note that each region may be multiply connected -that is, it may be made up of several disjoint regions. The boundaries between the regions Ω_i are the *Decision Boundaries*, or *Decision Surfaces*.

In the next sections we will assume the knowledge of the underlying class-conditional probability functions (the probability density functions of the feature vectors for a given class). Of course in many applications these will usually be unknown and must be estimated from a set of correctly classified samples termed the *training set*. In the following sections we will see how to construct *decision rules* that make use of available data in order to estimate the decision boundaries directly, without explicit calculation of the probability density functions.

2.2 The Bayes optimal solution

Consider C classes $\omega_1, \ldots, \omega_C$ with *a priori* probabilities, assumed known, $p(\omega_1), \ldots, p(\omega_C)$ (where $p(\omega_i)$ is the probability for the class ω_i to occur). If we wish to minimize the probability of making an error, then we would simply assign an object to class ω_i if

$$p(\omega_i) > p(\omega_j) \ \forall i = 1, \ldots, C \quad i \neq j$$

that is, if we have no informations regrding the objects, we would classify them as belonging to just one class (the most likely). For classes with equal probabilities, patterns would be assigned arbitrarily between those classes. Let us suppose that we have an observation vector $\mathbf{x}$ that we want to assign to one of the C classes. A *decision rule* based on probabilities is to assign $\mathbf{x}$ to class ω_i if the probabilities of class ω_i given the observation $\mathbf{x}$, $p(\omega_i|\mathbf{x})$, is greater then any other conditional probability for the other classes. That is, the rule will be:

$$\textit{assign } \mathbf{x} \textit{ to class } \omega_i \textit{ if}$$

$$p(\omega_i|\mathbf{x}) > \mathbf{p}(\omega_\mathbf{j}|\mathbf{x}) \ \forall \mathbf{i} = \mathbf{1}, \ldots, \mathbf{C} \ \mathbf{i} \neq \mathbf{j}$$

The *a posteriori* probabilities $p(\omega_i|\mathbf{x})$ may be expressed in terms of the *a priori* probabilities and the class conditional density functions $p(\mathbf{x}|\omega_i)$ using the *Bayes Theorem*:

$$p(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)p(\omega_i)}{p(\mathbf{x})}$$

Now the decision rule can be written this way:

$$\textit{assign } \mathbf{x} \textit{ to } \omega_i \textit{ if}$$

$$p(\mathbf{x}|\omega_i)p(\omega_i) > p(\mathbf{x}|\omega_j)p(\omega_j) \ \forall j = 1, \ldots, C \ i \neq j$$

(*Bayes Rule*). If we have only two classes, the former decision rule may alternatively be written:

$$if\ L_R(\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} > \frac{p(\omega_2)}{p(\omega_1)}$$

$$\textit{then } \mathbf{x} \textit{ belongs to class } \omega_1$$

Function $L_R(\mathbf{x})$ is called *Likelyhood ratio.* As an example, consider a two class problem with the class probability distribution showed in fig **??**.

Class ω_1 is normally distributed with zero mean and variance $\sigma = 1$, while class ω_2 is a weighted sum of normal densities. The *priors* are $p(\omega_1) = p(\omega_2) = 0.5$. If we plot the likelyhood function (see fig 2.2), it is clear how the decision rule partitions the measurement space into C multiply connected regions.

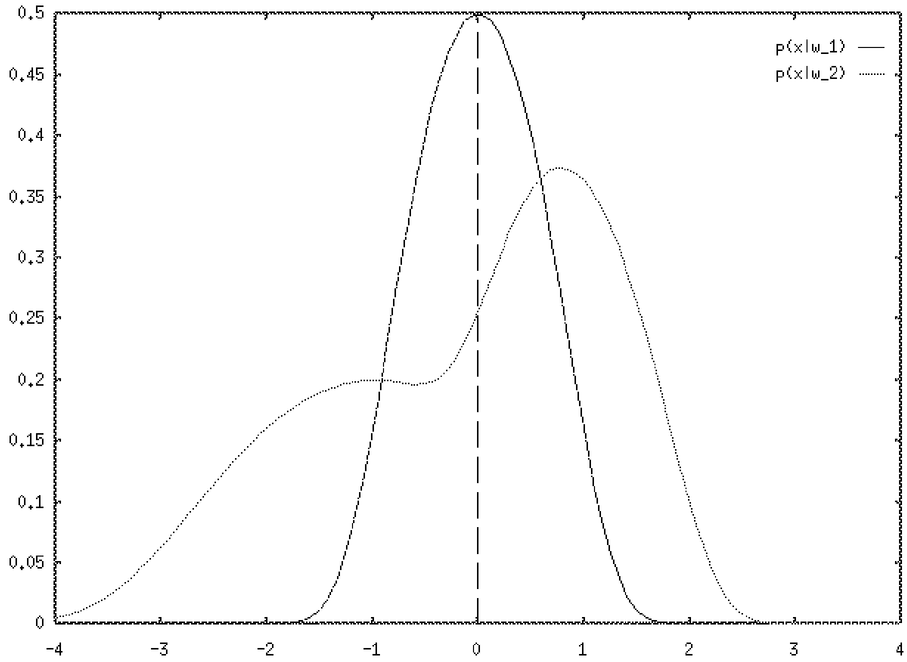

Figure 2.1: Example of two distributions

2.3 Discriminant Functions

Applying the Bayes Decision Rule, in order to achieve the proper classification, requires knowledge of the class-conditional density functions, $p(\mathbf{x}|\omega_i)$. In the literature there are many different density estimation methods useful for extracting informations from the data about the underlying statistical distributions. But, instead of making assumptions about $p(\mathbf{x}|\omega_i)$, we want to make assumptions about the form of the *discriminant functions*.
A *discriminant function* is a function of the pattern $\mathbf{x}$ that leads to a classification rule. For example, an optimal discriminant function for the two-class case that we have seen in the last section is

$$h(\mathbf{x}) = \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)}$$

where the rule is:

$$\begin{aligned} &if \;\; h(\mathbf{x}) > k \;\; then \;\; \mathbf{x} \in \omega_1 \\ &if \;\; h(\mathbf{x}) < k \;\; then \;\; \mathbf{x} \in \omega_2 \end{aligned}$$

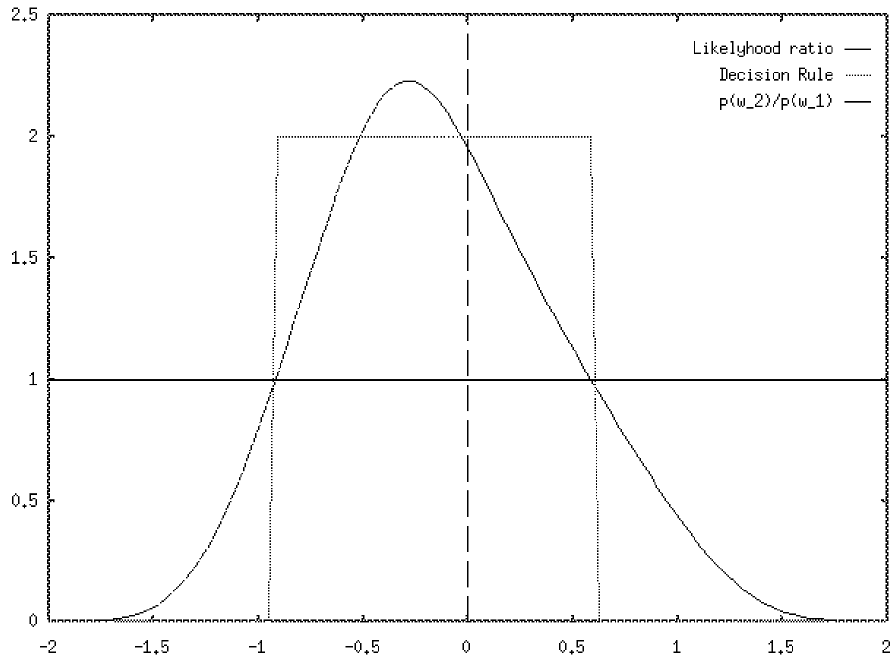

Figure 2.2: Resulting likelyhood function

with $k = \frac{p(\omega_2)}{p(\omega_1)}$.

2.3.0.0.1 Note *Discriminant functions are not unique. If f is a monotonic function, then the rule:*

$$if \quad g(\mathbf{x}) = f(h(\mathbf{x}) > k' \quad then \quad \mathbf{x} \in \omega_1$$
$$if \quad g(\mathbf{x}) = f(h(\mathbf{x}) < k' \quad then \quad \mathbf{x} \in \omega_2, \ with \quad k' = f(k).$$

leads to the same decision rule saw before.
The difference between the *Bayesian Approach* and the *discriminant function approach* described here is that the form of the discriminant function is specified and is not imponed by the underlying distribution. The choice of discriminant functions may depend on prior knowledge about the patterns to be classified, or may be a particular functional form whose parameters are adjusted by a *training* procedure. In the literature have been proposed different forms of discriminant functions, varying in complexity from the linear ones (in which g is a linear combination of the x_i) to non-linear parametric functions like classical multilayer perceptron. Discrimination may also be viewed as a problem in *regression*, in which the dependent variable, y, could be a class indicator, and the regressors are

the pattern vectors. In the following chapters we will introduce with more details the relationship between *regression* and *discrimination*.

2.4 Linear Discriminant Functions

In this section we will consider the discriminant functions defined as a linear combination of the components of $\mathbf{x} = (x_1, \ldots, x_n)$,

$$g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0 = \sum_{i=1}^{n} w_i x^i + w_0 \tag{2.1}$$

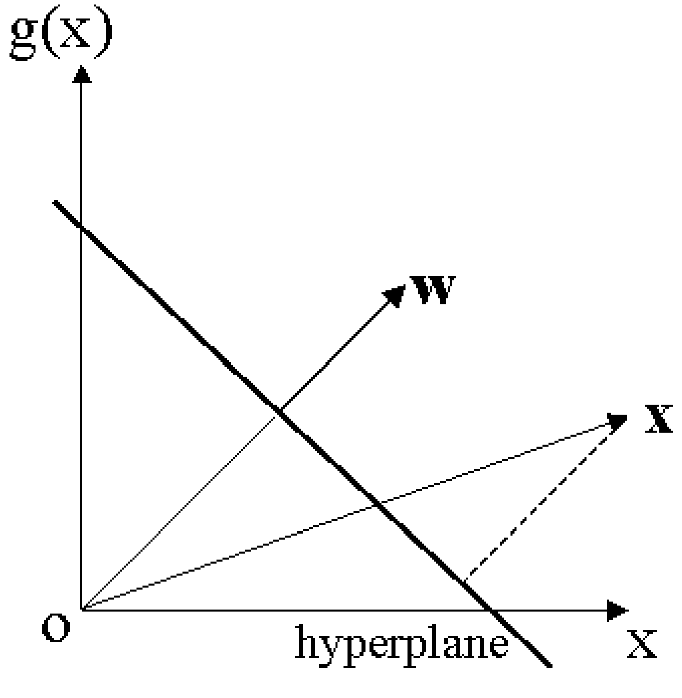

Figure 2.3: linear discriminant function

This is a *linear discriminant function* and is completely described by the (*weight*) vector $\mathbf{w}$ and w_0 (sometimes called *threshold vector*). Equation 2.1 describe an hyperplane with unit normal in the direction of $\mathbf{w}$ and whose perpendicular distance from origin is $\frac{|w_0|}{|\mathbf{w}|}$. The value of the discriminant function computed for a pattern $\mathbf{x}$ is the measure of the perpendicular distance from the hyperplane (see fig 2.3).

A linear discriminant function can arise trhough assumptions of normal distributions for class densities, with equal covariance matrices. Without making any distributional assumptions, we may impose the discriminant funcion to be linear and determine its parameters from considerations on the training set.

Given a linear classifier (also called *linear machine* by Nilsson [?]), and a set of patterns belonging to class $1, \ldots, C$, we now have to find out a rule for constructing the weight vector $\mathbf{w}$. Thus, let us consider a set of *prototype* points $\{\mathbf{p_1}, \ldots, \mathbf{p_C}\}$, one for each class $\omega_1, \ldots, \omega_C$. A simple and effective rule for creating a linear classifier up from these points is the *minimum instance rule* (or *nearest neighbour rule*). the resulting classifier will assign a pattern $\mathbf{x}$ to the class ω_i associated with the nearest point $\mathbf{p_i}$. The Euclidean squared distance of each point from the considered pattern $\mathbf{x}$ is :

$$|\mathbf{x} - \mathbf{p_i}|^2 = \mathbf{x} \cdot \mathbf{x} - 2\mathbf{x} \cdot \mathbf{p_i} + \mathbf{p_i} \cdot \mathbf{p_i}$$

thus the minimum distance classification is achieved by choosing the point $\mathbf{p_i}$ for which the expression $\mathbf{x} \cdot \mathbf{p_i} - 1/2\mathbf{p_i} \cdot \mathbf{p_i}$ assumes the largest value. The resulting linear discrimination function is

$$g_i(\mathbf{x}) = \mathbf{p_i} \cdot \mathbf{x} - \frac{1}{2}|\mathbf{p_i}|^\mathbf{2}$$

Each boundary is the perpendicular bisection of the lines joining the points of regions that are contiguous as showed in fig 2.4.

There are some problems in the linear discriminant approach we described so far. first, we have to remark that the decision regions are convex. This is a property of any linear machine, but in practice there are many two-class problems that, although separable, cannot be separated by a linear machine (see fig 2.5).

Moreover it is often the substitution of the points of a class with a unique, significant prototype point, also when class distribution is convex and linearly separable.

2.5 Generalized linear discriminant functions

A generalisation that overcomes the problems generated by non-linearly separable points sets is the so called *phi machine* (see Nillson 1965), a generalised linear discriminant function. The *phi machine* is a discriminant function of the form

$$g(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + w_0$$

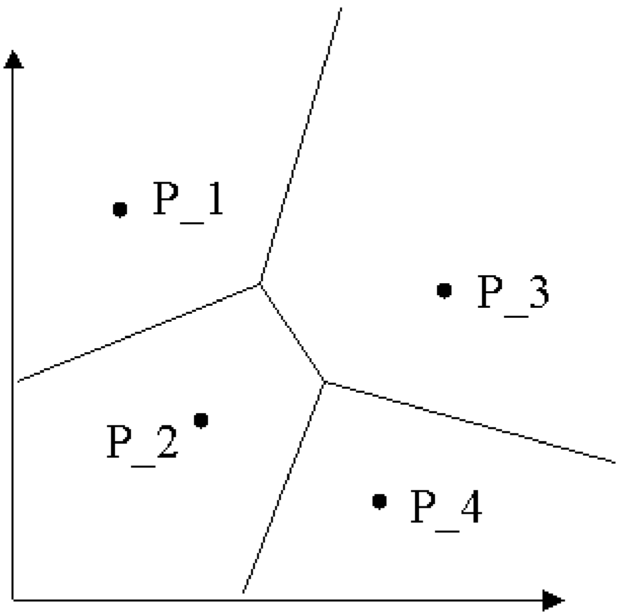

Figure 2.4: plane tassellation induced by minimum distance rule

where $\phi = (\phi_1(\mathbf{x}), \ldots, \phi_D(\mathbf{x}))$ is a vector function of $\mathbf{x}$. If $D = n$ and $\phi_i(\mathbf{x}) = x_i$, then we have a linear discriminant function. Let us note that such a discriminant function is linear in the function ϕ_i, not in the original measurements x_i.

For example, consider the two class problem of fig. 2.6; it is impossible to linearly separate the two distributions, even tought they are separable. However, if we apply the non-linear transformation

$$\phi_1 = x_1^2$$
$$\phi_2 = x_2$$

then the points will be projected in the ϕ space in such a way that now there exists a linearly separating hyperplane. The same hyperplane, expressed as a function of original coordinates x_1, x_2, describe a non-linear discriminant function.

Obviously in this formulation there is a problem in that, the more functions are used as a basis set, then the more parameters that must be determined using the same training set.

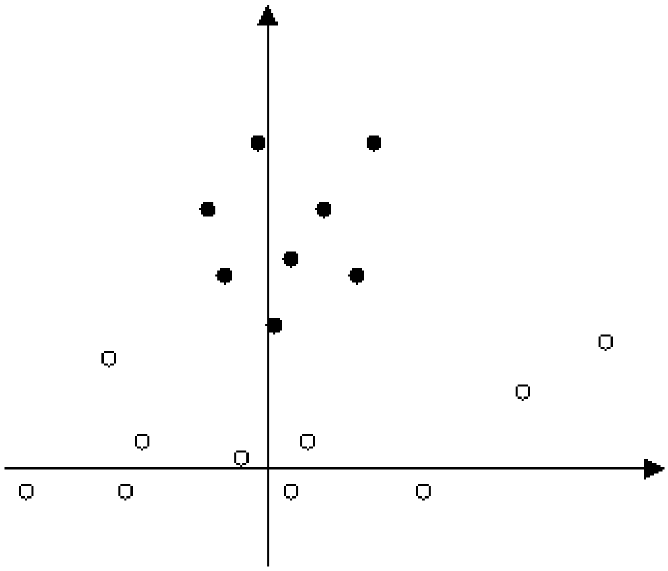

Figure 2.5: Example of non-linearly separable two-class problem

An alternative to having a set of different functions is to have a set of functions of the same parametric form, but which differ in the values of the parameters they have.

$$\phi_i(\mathbf{x}) = \phi(\mathbf{x}; v_i) \quad ,$$

where v_i is a set of parameters. Depending on the way the variable $\mathbf{x}$ and the parameters v_i interact we will obtain different models. For example if we define:

$$\phi(\mathbf{x}; v_i) = \phi(|\mathbf{x} - \mathbf{v}|)$$

then the resulting discriminant function is known as a *radial basis function*. In the next chapters we will see how to obtain non linear discriminant function s without having to specify such a set of parameters.

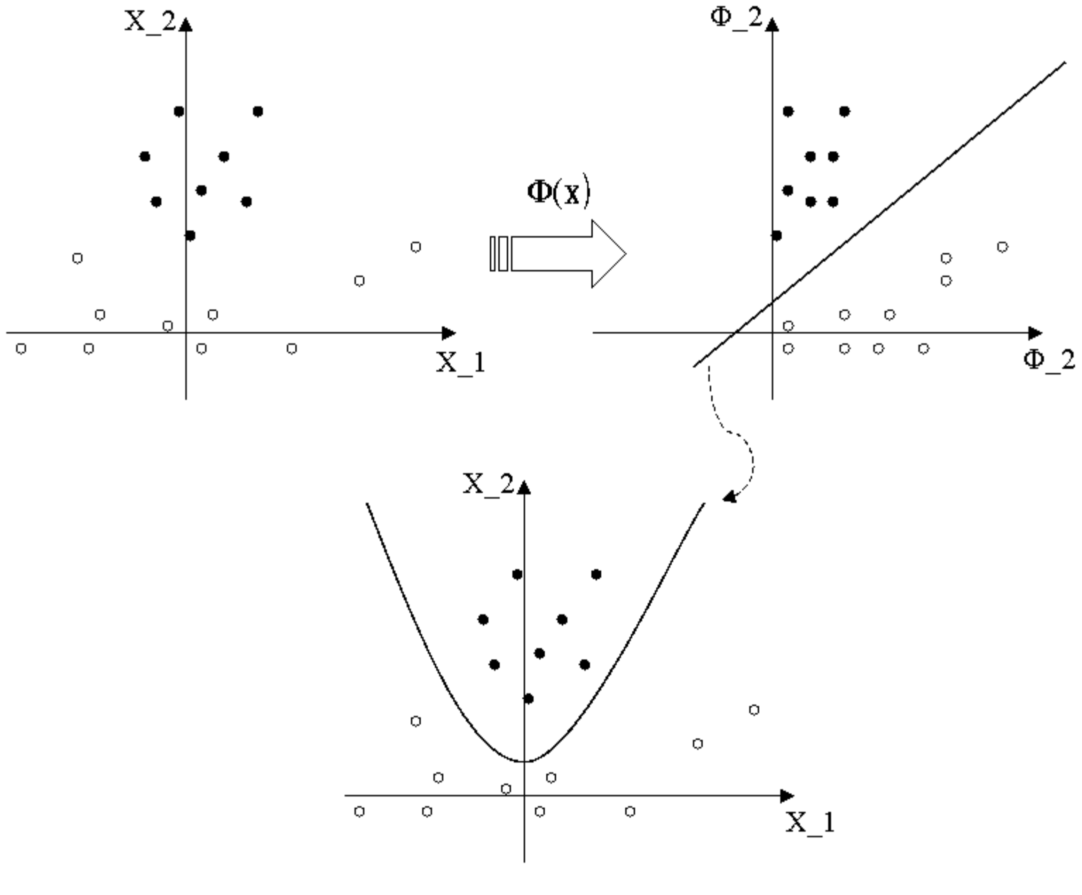

Figure 2.6: Non linear discriminant function

Chapter 3

Risk Minimization Principles

3.1 Statistical Pattern Recognition

Statistical Learning assumes that the learning process can be represented by the three-part scheme depicted in fig. 3.1.

As we can see in fig. 3.1, the process consists of:

- a *Pattern Generator*, able to choose a pattern according to a fixed unknown probability distribution $p(\mathbf{x})$
- a *Supervisor*, whose task is to associate a value y to a pattern $\mathbf{x}$ following a class-conditional density $p(y|\mathbf{x})$
- a *Learning Machine* (LM), who implements a function $f(\mathbf{x}, \alpha)$, described by a set of parameters $\alpha \in \Lambda$.

The purpose of the LM is to select among the possible functions, the one that best mimics the *supervisor*'s behaviour on the training set. The simplest and most intuitive way to guide the LM in its choice is by judging its performances on the training set. For this method being satistfied, we have to do some elementary hypothesis about the distribution type and shape. Then we have to choose among the available distributions the one that best fits experimental data.
As we have seen in the previous chapter, in order to evaluate the goodness of the discriminant function, it is customary to *weigh* each misclassification happened of the training phase by using a suitable *loss function*. In the simplest case the loss function *counts* the number of *errors* (misclassification). The expected value of the loss function over the train-

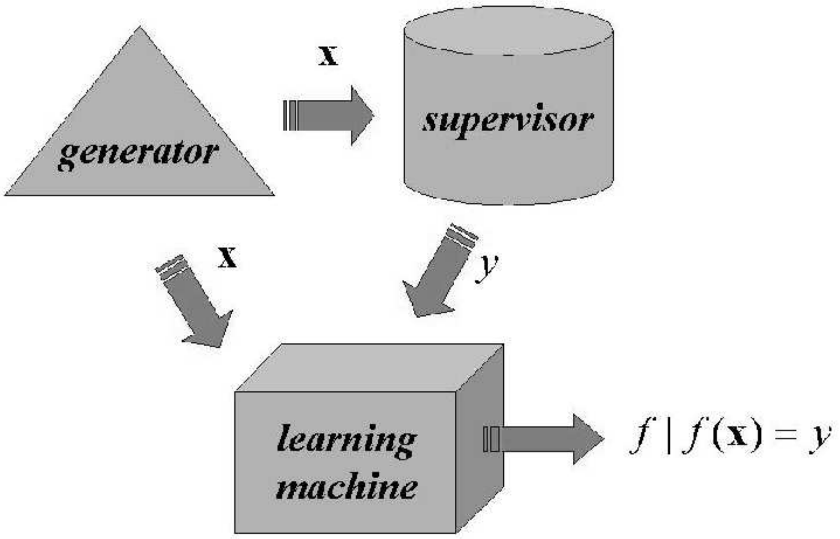

Figure 3.1: Sketch of the learning Process

ing set is called *Expected Risk*, and the *Empirical Risk Minimization principle* (ERM) aims at its minimization, as we will see in the following section.

3.2 ERM Principle

Formally, we define the *Expected Risk* the quantity:

$$R(\alpha) = \int Q(\mathbf{z}, \alpha) dP(\mathbf{z}) \tag{3.1}$$

whith

$$Q(\mathbf{z}, \alpha) dP(\mathbf{z}) = L(y, f(\mathbf{x}, \alpha)) dP(\mathbf{x}, y)$$

Let us remind that $P(\mathbf{x}, y) = P(\mathbf{x}) P(y|\mathbf{x})$ is the class conditional probability distribution, and $L(y, f(\mathbf{x}, \alpha))$ is the *Loss Function*. In a practical situation, however, we do not know the probability distribution $P(\mathbf{x}, y)$, since the only available information is represented by a set of experimental data points $\mathbf{z}_1 \ldots \mathbf{z}_l$ (where $\mathbf{z}$ denotes the pair $(\mathbf{x}, y)$). The general inductive principle called *Empirical Risk Minimization* aims at minimizing a discrete, empirical version of the *expected risk* defined in eqn. 3.1:

$$R_{emp}(\alpha) = \frac{1}{l} \sum_{i=1}^{l} Q(\mathbf{z}_i, \alpha) \tag{3.2}$$

The function for which $R_{emp}(\alpha)$ takes its minimum value is an approximation of the function that minimizes the functional 3.1.

3.2.1 Convergence and Consistence

The fundamental problem of Empirical Risk Minimization principle is to decide if and when this principle is *consistent*.

3.2.1.0.2 Definition Let us indicate with $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ a set of indicator functions. We say that ERM Principle is *not-trivially consistent* for the of functions $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ and the probability distribution $F(\mathbf{z})$, if for any nonempty subset $\Lambda(c), c \in (-\infty, \infty)$, defined as:

$$\Lambda(c) = \{\alpha : \int Q(\mathbf{z}, \alpha) dF(\mathbf{z}) > c, \alpha \in \Lambda\},$$

the following convergence in probability:

$$\inf_{\alpha\in\Lambda(c)} R_{emp}(\alpha) \xrightarrow{l\to\infty} \inf_{\alpha\in\Lambda(c)} R(\alpha)$$

is valid.
This convergence means that the ERM Principle is *not-trivially consistent* if it provides convergence for both the given set of functions and for all the subsets $\Lambda(c)$ of functions whose risk is bigger than c.

3.2.1.1 A note about triviality

For the ERM Principle be just *consistent* it suffices that the following convergences in probability are valid:

$$R_{emp}(\alpha) \xrightarrow{l\to\infty} \inf_{\alpha\in\Lambda(c)} R(\alpha) \tag{3.3}$$

$$R(\alpha) \xrightarrow{l\to\infty} \inf_{\alpha\in\Lambda(c)} R(\alpha) \tag{3.4}$$

This means that the learning process is consistent if both the expected risk $R(\alpha)$ and the empirical risk $R_emp(\alpha)$ converge to the minimal possible value of the risk, $\inf_{\alpha\in\Lambda(c)} R(\alpha)$ (see fig. 3.2).

Equation 3.3 states that the limit of a sequence of empirical risks gives the minimal possible value for the risk, while equation 3.4 asserts that the sequence of achieved risks converges (in probability) to the smallest possible risk for the given set of functions.
Unfortunately, for the above definition of consistency there are trivial cases that do not allow us to effectively generalize the consistency of ERM Principle. In order to provide an example of such a trivial case, let us consider a set of functions $Q(\mathbf{z},\alpha), \alpha\in\Lambda$ for which the ERM principle is not consistent (see fig. 3.3).

It is clear that if we add to the current set of functions a new function $\phi(\mathbf{z})$ such that:

$$\inf_{\alpha\in\Lambda(c)} Q(\mathbf{z},\alpha) > \phi(\mathbf{z})$$

and consider this new set of functions, then we will make the ERM Principle consistent. It is a trivial case of consistency, however, since the minimum of empirical risk is attained at the function $\phi(\mathbf{z})$ for any distribution function and for any number of observations.

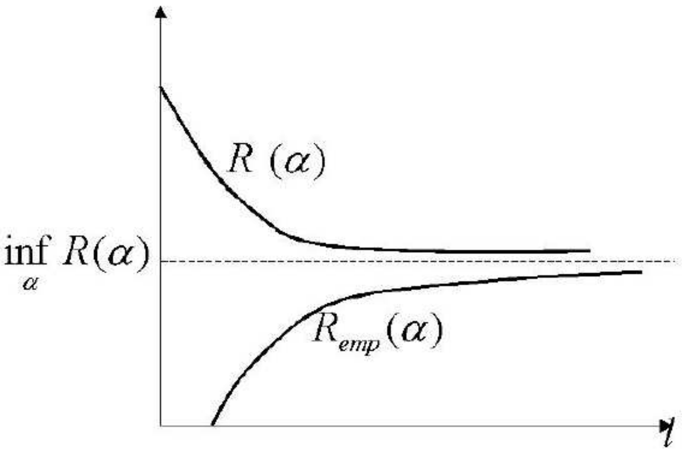

Figure 3.2: Simple consistence of ERM Principle

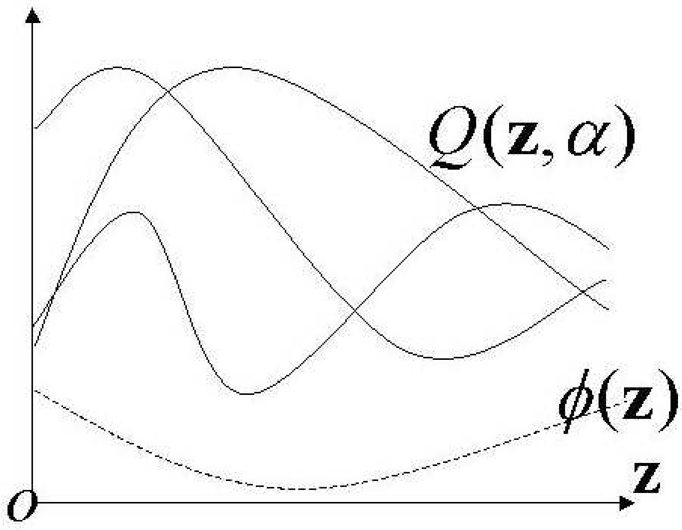

Figure 3.3: Example of trivial consistency

3.2.1.2 Uniform Convergence of ERM Principle

Let the probability function $F(\mathbf{z})$ be defined on the space $\mathbf{z} \in R^n$, and let $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ be a set of measurable (with respect to this distribution) functions.
Let $\mathbf{z}_1, \ldots, \mathbf{z}_l$ be a sequence of *i.i.d.* (*independent identically distributed*) vectors. The analysis of consistency of the ERM Principle, as we shall see, is connected with the analysis of the behaviour of an empirical process. The problem is to describe the conditions such that for any $\epsilon > 0$ the following convergence takes place:

$$P\{\sup_{\alpha \in \Lambda} |\int Q(\mathbf{z}, \alpha) dP(\mathbf{z}) - \frac{1}{l} \sum_{i=1}^{l} Q(\mathbf{z}_i, \alpha)| > \epsilon\} \stackrel{l \to \infty}{\longrightarrow} 0 \tag{3.5}$$

This relation indicates a *uniform convergence*, and it means that for sufficiently large l, the empirical risk functional approximates the risk functional uniformly well over all functions in a given set of functions. When uniform convergence takes place, the function which minimizes the empirical risk provides the value of the risk that is close to the smallest possible risk. Therefore the uniform convergence gives sufficient conditions for the consistency of the Empirical Risk Minimization Principle.
In order to characterize the relationship between uniform convergence and a qualitative (and quantitative) description of the capacity of a set of functions $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$, we are going to introduce a new concept: the *entropy* of a set of indicator functions.

3.3 Entropy and Capacity of a set of Functions

3.3.1 The error-event

Let us introduce the concept of *error-event*, defined as:

$$A_\alpha = \{\mathbf{z} : Q(\mathbf{z}, \alpha) > 0\}, \alpha \in \Lambda$$

In fact, we can think of the *error-event* as the samples that are not classified correctly by the particular function selected by the parameter set α. Roughly speaking, we will construct a measure of the capacity of the set of functions based on these *error-events*. For sake of semplicity, in the next section we will address the case of *finite* number of events, then we shall dedicate to the *infinite events* case.

3.3.2 Finite set of functions

By first we deal with the case of a finite set of indicator functions $Q(\mathbf{z}, \alpha_k), k = 1, \ldots, N$. In this case uniform convergence clearly holds. Indeed our set of error-events contains a finite number of elements:

$$A_k = \{\mathbf{z} : Q(\mathbf{z}, \alpha_k) > 0 \quad k = 1, \ldots, N\}$$

and we can say :

$$P\{\max_{1 \leq k \leq N} |P\{Q(\mathbf{z}, \alpha_k) > 0\} - \phi_l\{Q(\mathbf{z}, \alpha_k) > 0\}| > \epsilon\} \leq 2e^{(\frac{lnN}{2} - 2\epsilon^2)l}$$

where $\phi_l\{Q(\mathbf{z}, \alpha_k) > 0\}$ denotes the frequency of event over a sample of size l.
In order to achieve uniform convergence for any ϵ the limit:

$$\frac{lnN}{l} \overset{l \to \infty}{\longrightarrow} 0 \tag{3.6}$$

has to be valid. This proves that if N is finite the ERM is consistent, but also gives us an indication about what could happen if the number of indicator functions is infinite.
Moreover equation 3.6 also provides first estimate of the *capacity* of the set of functions. In the next section we will see how to generalize this *capacity* concept.

3.3.3 Infinite set of functions

Let us begin by observing that only a finite number of cluster of error-events is distinguishable on the given set $\mathbf{z}_1, \ldots, \mathbf{z}_l$ (two error-events are distinguishable if there exists at least one element in the sample that belongs to one event and does not belongs to the other).
We shall denote $N^\Lambda(\mathbf{z}_1, \ldots, \mathbf{z}_l)$ the number of clusters. The value of $N^\Lambda(\mathbf{z}_1, \ldots, \mathbf{z}_l)$ depends on the sample $\mathbf{z}_1, \ldots, \mathbf{z}_l$ and on the set of error-events $A_\alpha = \{\mathbf{z} : Q(\mathbf{z}, \alpha) > 0\}, \ \alpha \in \Lambda$. It is possible to show that if the value of N^Λ increases slowly as the sample size l increases, then uniform convergence takes place.
More formally, consider a set of indicator functions $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ on the set $\mathbf{Z}$, and an arbitrary sequence of l vectors from $\mathbf{Z}$: $\mathbf{z}_1, \ldots, \mathbf{z}_l$.
We can determine the set of l-dimensional binary vectors

$$q(\alpha) = (Q(\mathbf{z}_1, \alpha), \ldots, Q(\mathbf{z}_1, \alpha)), \alpha \in \Lambda.$$

For any fixed α, this vector determines a collection of vertices of a unit l-dimensional cube (see fig. 3.4)

Now, let us denote the number of different vertices with

$$N^\Lambda(\mathbf{z}_1, \ldots, \mathbf{z}_l)$$

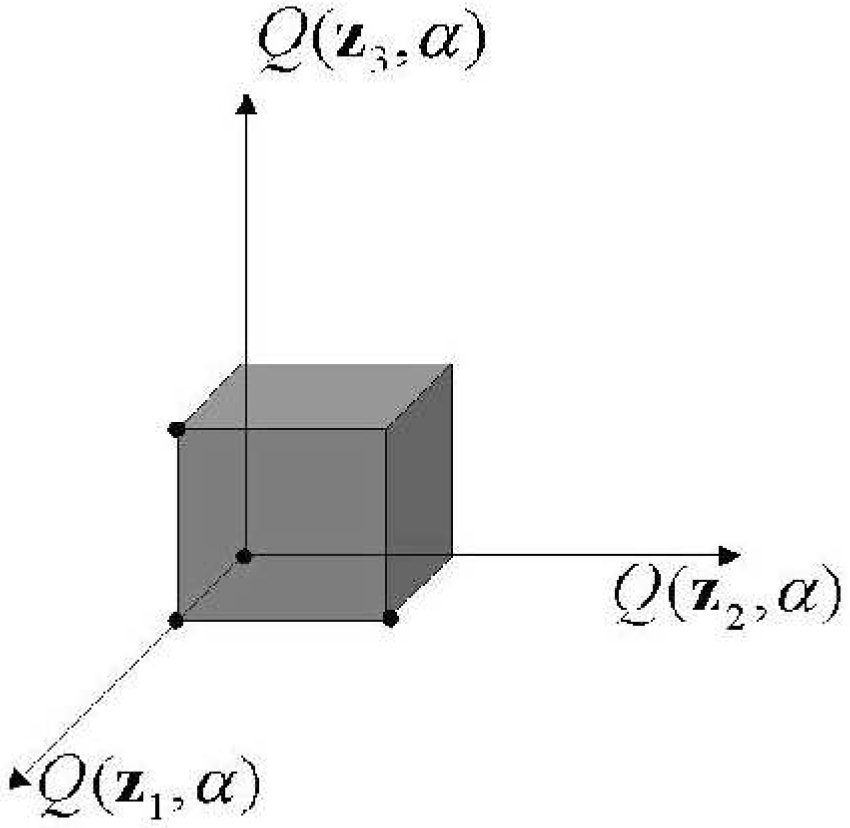

Figure 3.4: Unit cube in the l-dimensional space

This number depends both on the sample and on the set of functions, and is clearly bounded by:

$$N^{\Lambda}(\mathbf{z}_1, \ldots, \mathbf{z}_l) \leq 2^l$$

3.3.3.0.1 Definition We say that the quantity

$$H^{\Lambda}(\mathbf{z}_1, \ldots, \mathbf{z}_l) = ln N^{\Lambda}(\mathbf{z}_1, \ldots, \mathbf{z}_l)$$

is the **random entropy** of the set of indicator functions $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ on the sample $\mathbf{z}_1, \ldots, \mathbf{z}_l$. Moreover we define **entropy** of the set of functions over a sample of size l the quantity:

$$H^{\Lambda}(l) = \int H^{\Lambda}(\mathbf{z}_1, \ldots, \mathbf{z}_l) dF(\mathbf{z}_1, \ldots, \mathbf{z}_l)$$

The first result that binds together *entropy* and uniform convergence is the following theorem;

Theorem 1 *In order that uniform convergence over the set of indicator functions* $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ *takes place, it is necessary and sufficient that:*

$$\frac{H^{\Lambda}}{l} \stackrel{l \to \infty}{\longrightarrow} 0$$

be valid.

Let us remark that along with the concept of *entropy* we introduced a new "tool" for characterizing the capacity of a set of functions, as we did in the simpler case of finite number of error-events.
The *entropy* completely defines the *qualitative* behaviour of the learning process: the consistency of learning
Now we are interested in the *quantitative* behaviour of the process, in order to give a complete characterization of the ERM Principle. In other words we want to know under which conditions, for any probability measure $F(\mathbf{z})$, the asymptotic rate of uniform convergence is "fast" enough (that is with exponential bounds).

3.3.4 The Growth Function

For giving an answer to this question, let us introduce a new concept based on the value of $N^{\Lambda}(\mathbf{z}_1, \ldots, \mathbf{z}_l)$, the *Growth Function* $G^{\Lambda}(l)$, defined as follows:

$$G^{\Lambda}(l) = ln \ \sup_{\mathbf{z}_1, \ldots, \mathbf{z}_l} N^{\Lambda}(\mathbf{z}_1, \ldots, \mathbf{z}_l)$$

Now we are able of describing the *necessary and sufficient* condition for consistency of ERM for any probability measure: the following assertion

$$\lim_{l \to \infty} \frac{G^{\Lambda}(l)}{l} = 0 \tag{3.7}$$

has to be verified.

Note 1 *Equation 3.7 is a significant result, since it describes the necessary and sufficient conditions under which a learning machine that implements the ERM Principle has high asymptotic rate of convergence, independently of the probability measure, and therefore independent of the particular problem to be solved.*

3.3.5 The VC-Dimension

Using the *growth function*, that we have defined in the previous chapter, it is possible to obtain some main bounds on the generalization ability of learning machines. Let us start by noting that, while the *growth function* has the good property of being independent from the probability measure, the bounds based on it are unfortunately *non-constructive*, since the theory does not suggest how to evaluate the *growth function* for a given (possibly infinite) set of functions. Thus, in this section, we will consider another (more useful) estimate of the generalization ability of a learning machine based on the *growth function*.

Theorem 2 *The* growth function *of a set of indicator functions* $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ *either*

- *satisfies the equality:*

$$G^{\Lambda}(l) = ln\ 2$$

- *or is bounded by:*

$$G^{\Lambda}(l) \begin{cases} = ln\ 2 & if\ \ l \leq h \\ \leq h(1 + ln(\frac{l}{h}) & if\ \ l > h \end{cases}$$

See fig. 3.5

Definition 1 *The capacity of a set of functions with logarithmic bounded growth functions can be characterized by the coefficient* **h***.*
h *is called "***VC-Dimension** *of a set of indicator functions". When the* growth function *is linear the* VC-dimension *is defined to be infinite.*

More intuitively we can define the *VC-dimension* of a set of indicator functions $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ as the largest number h of vectors $\mathbf{z}_1, \ldots, \mathbf{z}_l$ that can be separated into two different classes in all the possible 2^h ways using this set of functions. For example, if we look at fig. 3.6, we can see that if our set of indicator functions is composed by straight lines, it is possible to separate at most three points in the plane in all the $2^3 = 8$ ways, while if we have 4 points there is at least a configuration that we can not obtain using straight lines. Therefore the *VC-dimension* of this set of functions in the 2-dimensional plane is 3.

There are several cases where is possible to calculate the *VC-dimension* of the set of indicator functions. In those cases, we can provide *constructive*, *distribution-free* bounds on the learning machine that minimizes the *empirical risk functional*:

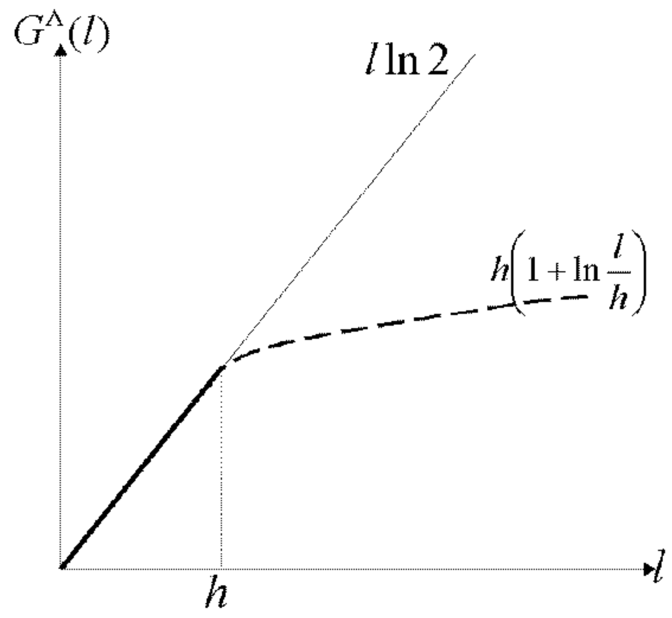

Figure 3.5: The VC-dimension is the value at which the growth function ceases to be linear

Theorem 3 *With probability* $1-\eta$ *the risk for the function* $Q(\mathbf{z},\alpha)$ *which minimizes the Empirical functional satisfies:*

$$R(\alpha_l) < R_emp(\alpha_l) + \frac{\epsilon(l)}{2}(1 + \sqrt{1 + \frac{4R_emp(\alpha_l)}{\epsilon(l)}}) \tag{3.8}$$

where

$$\epsilon(l) = 4\frac{h\ ln\ (\frac{2l}{h+1}) - ln(\frac{\eta}{4})}{l}$$

In addition to this result, the finiteness of the *VC-dimension* of a set of indicator functions $Q(\mathbf{z},\alpha), \alpha \in \Lambda$ is a *necessary and sufficient* condition for the uniform convergence, and therefore for the consistency of the ERM Principle.

3.4 The SRM Principle

Very often the ERM Principle gives good results when applied to practical problems. But if we examine the bound 3.8 we see that this will happen only when the number of the

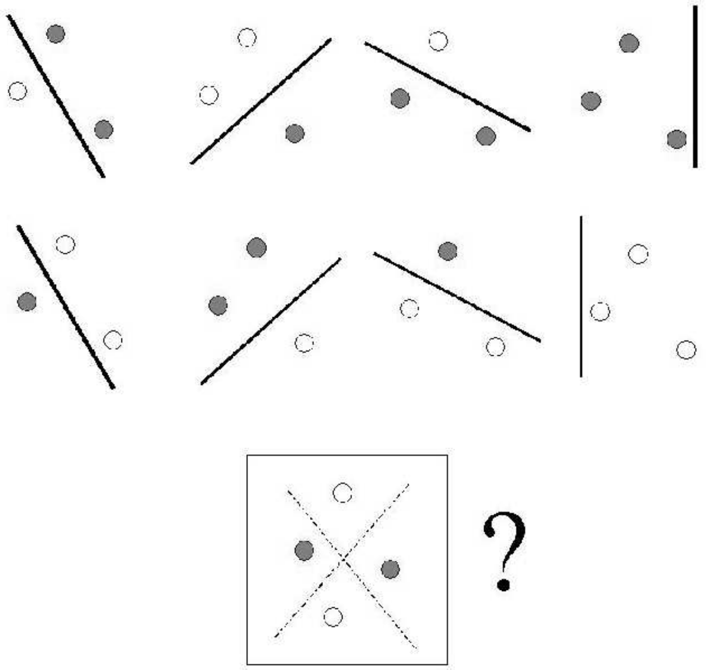

Figure 3.6: "Intuitive" example of VC-dimension: the set of straight lines in the 2-dim plane can *shatter* at most three points

available samples is high if compared with the *VC-dimension* of the considered set of functions. In that case, therefore, minimizing the *empirical risk* allows us to control over the whole learning process. But what if we do not have a large *training set* (say, for example, that $\frac{l}{h} < 20$)?
Again, equation 3.8 suggests that for achieving the smallest *actual risk* it is not sufficient to keep low the *empirical risk*, but it is necessary to minimize the right-hand side of inequality 3.8 over both terms simultaneusly. Since for a fixed number of observations, the second term of 3.8 depends mainly on the *VC-dimension* of the set of functions, we have to arrange the set of functions in such a way that their *VC-dimension* become a *controlling variable*.

Consider a structure S made up of nested sets S_i of functions $Q(\mathbf{z}, \alpha), \alpha \in \Lambda$ such that:

$$S_1 \subset S_2 \subset \dots$$

where

$$S_i = \{Q(\mathbf{z}, \alpha), \alpha \in \Lambda_i\} \quad and \quad S = \cup_i S_i$$

We will deal only with those structures S for which any element $S_i \in S$ has a finite *VC-dimension* h_i.
The most important property of structure S, which arises from its construction, is that

- The sequence of values of *VC-dimensions* h_i for the elements S_i of the structure S is nondecreasing with increassing h:

$$h_i \leq h_2 \leq \dots$$

Therefore, for a given set of observations, we call *Structural Risk Minimization Principle* (SRM) the method which chooses the element S_i of the structure for which the smallest guaranteed risk is achieved (see fig. 3.7)

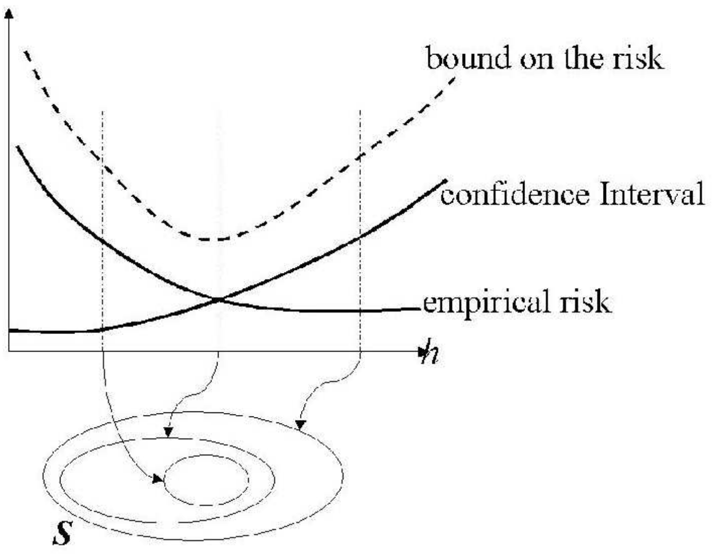

Figure 3.7: *Structural Minimization Principle* uses *VC-dimension* as a controlling function for achieving the smallest value of the *actual risk*.

This principle provides a powerful and general model of *capacity control*. It is also possible to prove that SRM Principle is always consistent, and defines a bound on the rate of convergence (For this and other proofs, the interested reader can see Vapnik '98).

Chapter 4

Support Vector Machines

In Chapter 1 we introduced the pattern recognition problem as a *discrimination* problem, in which some functions are used to partition the space of training data in order to construct the best *decision surfaces.Binary classification* may be considered as a special pattern recognition problem, in which we have to choose between just two classes. In this framework SVM's provide a new method for constructing linear and non-linear (in the sense of *phi-machine* that we introduced in Chapter 1) decision surfaces. In this chapter we introduce SVM for classification from the point of view of constrained optimization.

4.1 SVM for classification

Support Vector Machines for pattern classification have been introduced by Cortes and Vapnik in 1995. Consider the classical linear pattern recognition problem, that we approached in chapter 1 using Bayesian formalism. In the linear, separable case the key idea of a SVM can be explained in plain words. Given a training set T_l which contains points of either of two classes, a SVM separates the classes through a hyperplane determined by certain points of T_l, termed *support vectors*. In the separable case, this hyperplane maximizes the *margin*, or twice the minimum distance of either class from the hyperplane, and all the support vectors lie at the same minimum distance from the hyperplane (and are thus termed *margin vectors*). In real cases, the two classes may not be separable and both the hyperplane and the support vectors are obtained from the solution of a problem of constrained optimization. The solution is a trade-off between the largest margin and the lowest number of errors, trade-off controlled by a regularization parameter.

4.1.1 Optimal separating hyperplane

In what follows we assume we are given a set T_l of points $\mathbf{x}_i \in \Re^n$ with $i = 1, 2, \ldots, l$. Each point $\mathbf{x}_i$ belongs to either of two classes and thus is given a label $y_i \in \{-1, 1\}$. The goal is to establish the equation of a hyperplane that divides T_l leaving all the points of the same class on the same side while maximizing the minimum distance between either of the two classes and the hyperplane. To this purpose we need some preliminary definitions.

Definition 1. The set T_l is *linearly separable* if there exist $\mathbf{w} \in \Re^n$ and $b \in \Re$ such that

$$\begin{array}{ll} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 & \text{if } y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 & \text{if } y_i = -1. \end{array} \tag{4.1}$$

In more compact notation, the two inequalities (4.1) can be rewritten

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \tag{4.2}$$

for $i = 1, 2, \ldots, l$. The pair $(\mathbf{w}, b)$ defines a hyperplane of equation

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

named *separating hyperplane* (see figure **??**(a)). If we denote with w the norm of $\mathbf{w}$, the signed distance d_i of a point $\mathbf{x}_i$ from the separating hyperplane $(\mathbf{w}, b)$ is given by

$$d_i = \frac{\mathbf{w} \cdot \mathbf{x}_i + b}{w}. \tag{4.3}$$

Combining inequality (4.2) and equation (4.3), for all $x_i \in T_l$ we have

$$y_i d_i \geq \frac{1}{w}. \tag{4.4}$$

Therefore, $1/w$ is the lower bound on the distance between the points $\mathbf{x}_i$ and the separating hyperplane $(\mathbf{w}, b)$.

One might ask why not simply rewrite inequality (4.2) as

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 0.$$

The purpose of the "1" in the right hand side of inequality (4.2) is to establish a one-to-one correspondence between separating hyperplanes and their parametric representation. This is done through the notion of canonical representation of a separating hyperplane[1].

[1] This intermediate step toward the derivation of optimal separating hyperplanes is slightly different from the derivation originally developed in (Cortes and Vapnik 1995).

Definition 2 *Given a separating hyperplane* $(\mathbf{w}, b)$ *for the linearly separable set* T_l*, the* canonical representation *of the separating hyperplane is obtained by rescaling the pair* $(\mathbf{w}, b)$ *into the pair* $(\mathbf{w}', b')$ *in such a way that the distance of the closest point equals* $1/w'$*.*

Through this definition we have that

$$\min_{\mathbf{x}_i \in T_l} \{y_i(\mathbf{w}' \cdot \mathbf{x}_i + b')\} = 1.$$

Consequently, for a separating hyperplane in the canonical representation, the bound in inequality (4.4) is tight. In what follows we will assume that a separating hyperplane is always given the canonical representation and thus write $(\mathbf{w}, b)$ instead of $(\mathbf{w}', b')$. We are now in a position to define the notion of optimal separating hyperplane.

Definition 3 *Given a linearly separable set* T_l*, the* optimal separating hyperplane *(OSH) is the separating hyperplane which maximizes the distance of the closest point of* T_l*.*

Since the distance of the closest point equals $1/w$, the OSH can be regarded as the solution of the problem of maximizing $1/w$ subject to the constraint (4.2), or

Problem **P1**
Minimize $\frac{1}{2}\mathbf{w} \cdot \mathbf{w}$
subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \ldots, l$

Two comments are in order. First, if the pair $(\mathbf{w}, b)$ solves **P1**, then for at least one $\mathbf{x}_i \in T_l$ we have $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$. In particular, this implies that the solution of **P1** is always a separating hyperplane in the canonical representation. Second, the parameter b enters in the constraints but not in the function to be minimized.

The quantity $2/w$, which measures the distance between the two classes in the direction of $\mathbf{w}$, is named *margin*. Hence, the OSH can also be seen as a separating hyperplane which maximizes the margin (see figure 4.1).

Usually, the *VC-Dimension* of thes hyperplanes would be $n+1$. However, as the following theorem states (see Vapnik 98), if we construct a suitable *structure* on the hyperplane space, we can tune its vc dimension.

Theorem 4 *Given a set of canonical hyperplanes defined over a set of vectors*

$$X^* = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}, |\mathbf{x}_i - \mathbf{a}| \leq R$$

that satisfy the following condition

$$\|\mathbf{w}\| \leq A,$$

then we can say that for the VC-dimension of the set of hyperplanes h the following inequality

$$h \leq min(R^2A^2, n) + 1 \tag{4.5}$$

olds

Since for hypothesis we are dealing with a linearly separable problem, it is possible to lower to zero the *empirical* risk (that is classifying without any error the training set). Therefore minimizing the norm of $\mathbf{w}$ means is equivalent to minimizing the *VC-dimension* of the considered set of hyperplanes, searching for the solution which achieves the smallest structural risk.
We now study the properties of the solution of the problem **P1**.

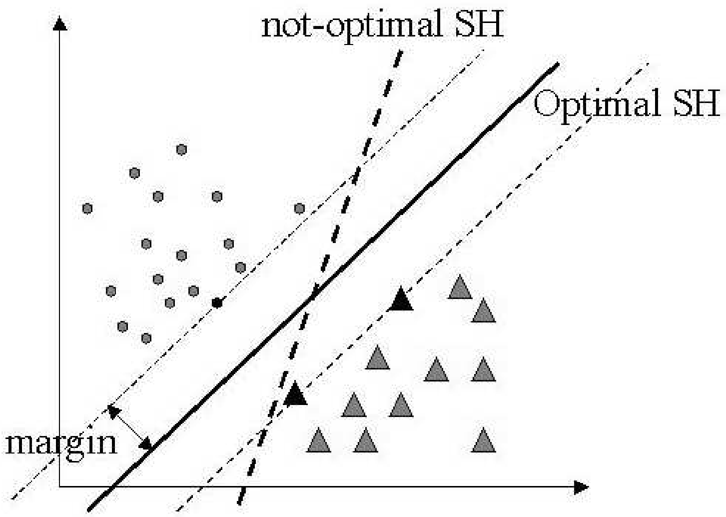

Figure 4.1: the Optimal Separating Hyperplane maximizes the margin between the sets of circles and triangles

4.1.2 Support vectors

Problem **P1** can be solved by means of the classical method of Lagrange multipliers (Bazaraa and Shetty 1979). If we denote with $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_l)$ the l nonnegative Lagrange multipliers associated with the constraints (4.2), the solution to problem **P1** is equivalent to determining the *saddle point* of the function

$$L = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^{l} \alpha_i \left\{ y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \right\}. \tag{4.6}$$

with L = L($\mathbf{w}$, b, $\boldsymbol{\alpha}$). At the saddle point, L has a minimum for $\mathbf{w} = \bar{\mathbf{w}}$ and $b = \bar{b}$ and a maximum for $\boldsymbol{\alpha} = \bar{\boldsymbol{\alpha}}$, and thus we can write

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l} y_i \alpha_i = 0, \tag{4.7}$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i = 0 \tag{4.8}$$

with

$$\frac{\partial L}{\partial \mathbf{w}} = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_n}).$$

Substituting equations (4.7) and (4.8) into the right hand side of (4.6), we see that problem **P1** reduces to the maximization of the function

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j,$$

subject to the constraint (4.7) with $\boldsymbol{\alpha} \geq 0$[2]. This new problem is called *dual problem* and can be formulated as

Problem **P2**

Maximize	$-\frac{1}{2}\boldsymbol{\alpha} \cdot D\boldsymbol{\alpha} + \sum \alpha_i$
subject to	$\sum y_i \alpha_i = 0$
	$\boldsymbol{\alpha} \geq 0,$

where both sums are for $i = 1, 2, \ldots, l$, and D is an $l \times l$ matrix such that

$$D_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{4.9}$$

As for the pair $(\bar{\mathbf{w}}, \bar{b})$, from equation (4.8) it follows that

$$\bar{\mathbf{w}} = \sum_{i=1}^{l} \bar{\alpha}_i y_i \mathbf{x}_i, \tag{4.10}$$

while $\bar{b}$ can be determined from the Kuhn-Tucker conditions

$$\bar{\alpha}_i \left(y_i(\bar{\mathbf{w}} \cdot \mathbf{x}_i + \bar{b}) - 1 \right) = 0, \qquad i = 1, 2, \ldots, l. \tag{4.11}$$

Note that the only $\bar{\alpha}_i$ that can be nonzero in equation (4.11) are those for which the constraints (4.2) are satisfied with the equality sign. The corresponding points $\mathbf{x}_i$, termed *support vectors*, are the points of T_l closest to the OSH (see figure **??**(b)).

Given a support vector $\mathbf{x}_j$, the parameter $\bar{b}$ can be obtained from the corresponding Kuhn-Tucker condition as

$$\bar{b} = y_j - \bar{\mathbf{w}} \cdot \mathbf{x}_j.$$

[2]In what follows $\boldsymbol{\alpha} \geq 0$ means $\alpha_i \geq 0$ for every component α_i of any vector $\boldsymbol{\alpha}$.

The problem of classifying a new data point $\mathbf{x}$ is now simply solved by computing

$$\text{sign}\left(\bar{\mathbf{w}} \cdot \mathbf{x} + \bar{b}\right). \tag{4.12}$$

In conclusion, the support vectors condense all the information contained in the training set T_l which is needed to classify new data points.

4.1.3 Linearly nonseparable case

If the set T_l is not linearly separable or one simply ignores whether or not the set T_l is linearly separable, the problem of searching for an OSH is meaningless (there may be no separating hyperplane to start with). Fortunately, the previous analysis can be generalized by introducing l nonnegative variables $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_l)$ such that

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \qquad i = 1, 2, \ldots, l. \tag{4.13}$$

If the point $\mathbf{x}_i$ satisfies inequality (4.2), then ξ_i is null and (4.13) reduces to (4.2). Instead, if the point $\mathbf{x}_i$ does not satisfy inequality (4.2), the term $-\xi_i$ is added to the right hand side of (4.2) to obtain inequality (4.13). The generalized OSH is then regarded as the solution to

Problem **P3**
Minimize $\quad \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C \sum \xi_i$
subject to $\quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, 2, \ldots, l$
$\quad \boldsymbol{\xi} \geq 0.$

The term $C \sum \xi_i$, where the sum is for $i = 1, 2, \ldots, l$, can be thought of as some measure of the amount of misclassification. Note that this term leads to a more robust solution, in the statistical sense, than the intuitively more appealing term $C \sum \xi_i^2$. In other words, the term $C \sum \xi_i$ makes the OSH less sensitive to the presence of outliers in the training set. The parameter C can be regarded as a regularization parameter. The OSH tends to maximize the minimum distance $1/w$ for small C, and minimize the number of misclassified points for large C. For intermediate values of C the solution of problem **P3** trades errors for a larger margin.

In analogy with what was done for the separable case, problem **P3** can be transformed into the *dual*

Problem **P4**
Maximize $\quad -\frac{1}{2}\boldsymbol{\alpha} \cdot D\boldsymbol{\alpha} + \sum \alpha_i$
subject to $\quad \sum y_i \alpha_i = 0$
$\quad 0 \leq \alpha_i \leq C, \qquad i = 1, 2, \ldots, l$

with D the same $l \times l$ matrix of the separable case. Note that the dimension of **P4** is given by the size of the training set, while the dimension of the input space gives the rank of D. From the constraints of problem **P4** it follows that if C is sufficiently large and the set T_l linearly separable, problem **P4** reduces to **P2**.

As for the pair $(\bar{\mathbf{w}}, \bar{b})$, it is easy to find that

$$\bar{\mathbf{w}} = \sum_{i=1}^{l} \bar{\alpha}_i y_i \mathbf{x}_i,$$

while $\bar{b}$ can again be determined from $\bar{\boldsymbol{\alpha}}$, solution of the dual problem **P4**, and from the new Kuhn-Tucker conditions

$$\bar{\alpha}_i \left(y_i (\bar{\mathbf{w}} \cdot \mathbf{x}_i + \bar{b}) - 1 + \bar{\xi}_i \right) = 0 \qquad (4.14)$$

$$(C - \bar{\alpha}_i) \bar{\xi}_i = 0 \qquad (4.15)$$

where the $\bar{\xi}_i$ are the values of the ξ_i at the saddle point. Similarly to the separable case, the points $\mathbf{x}_i$ for which $\bar{\alpha}_i > 0$ are termed *support vectors*. The main difference is that here we have to distinguish between the support vectors for which $\bar{\alpha}_i < C$ and those for which $\bar{\alpha}_i = C$. In the first case, from condition (4.15) it follows that $\bar{\xi}_i = 0$, and hence, from condition (4.14), that the support vectors lie at a distance $1/\bar{w}$ from the OSH. These support vectors are termed *margin vectors*. The support vectors for which $\bar{\alpha}_i = C$, instead, are misclassified points (if $\xi_i > 1$), points correctly classified but closer than $1/\bar{w}$ from the OSH (if $0 < \xi \leq 1$), or, in some degenerate cases, even points lying on the margin (if $\xi_i = 0$). In any event, we refer to all the support vectors for which $\alpha_i = C$ as *errors*. An example of generalized OSH with the relative margin vectors and errors is shown in figure 4.2. All the points that are not support vectors are correctly classified and lie outside the margin strip.

We now conclude this section by discussing the extension of the theory to the *nonlinear* case.

4.1.4 Nonlinear kernels

In most cases, linear separation in input space is a too restrictive hypothesis to be of practical use. As we have already seen in Chapter 1, a common way to overcome this limitation is using a *mapping* from the original training input space into another space in which, hopefully, the discrimination problem becomes *linear*. Fortunately, the SVM theory can be extended to nonlinear separating surfaces by mapping the input points into feature points and looking for the OSH in the corresponding feature space (Cortes and Vapnik 1995).

If $\mathbf{x} \in \Re^n$ is an input point, we let $\boldsymbol{\varphi}(\mathbf{x})$ be the corresponding feature point with $\boldsymbol{\varphi}$ a mapping from $\Re^n$ to a certain space Z (typically a Hilbert space of finite or infinite dimension). In both cases we denote with φ_i the components of $\boldsymbol{\varphi}$. Clearly, to an OSH in Z corresponds a nonlinear separating surface in input space.

At first sight it might seem that this nonlinear surface cannot be determined unless the mapping $\boldsymbol{\varphi}$ is completely known. However, from the formulation of problem **P4** and the classification stage of equation (4.12), it follows that $\boldsymbol{\varphi}$ enters only in the dot product between feature points, since

$$D_{ij} = y_i y_j \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j),$$

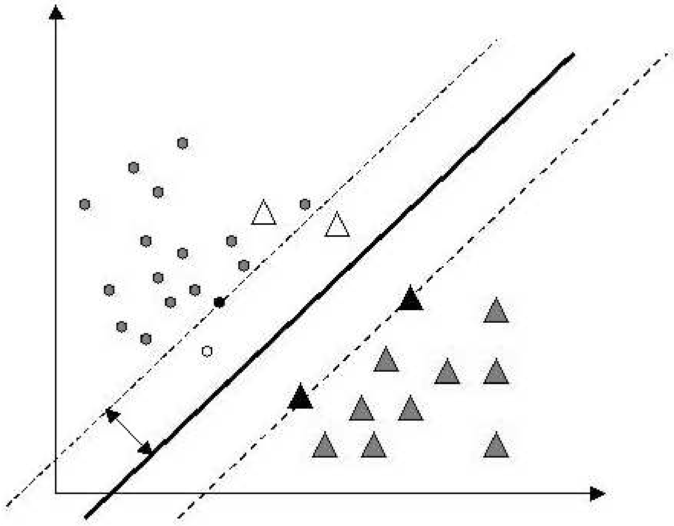

Figure 4.2: Example of OSH in case of misclassifications

and

$$\bar{\mathbf{w}} \cdot \boldsymbol{\varphi}(\mathbf{x}) + \bar{b} = \sum \bar{\alpha}_i y_i \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}) + \bar{b}.$$

Consequently, if we find an expression for the dot product in feature space which uses the points in input space only, that is

$$\boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j), \tag{4.16}$$

full knowledge of $\boldsymbol{\varphi}$ is not necessary. The symmetric function K in equation (4.16) is called *kernel*. The nonlinear separating surface can be found as the solution of problem **P4** with $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, while the classification stage reduces to computing

$$\text{sign}\left(\sum \bar{\alpha}_i y_i K(\mathbf{x}_i, \mathbf{x}) + \bar{b}\right).$$

Therefore, the extension of the theory to the nonlinear case is reduced to finding kernels which identify certain families of decision surfaces and can be written as in equation (4.16). A useful criterion for deciding whether a kernel can be written as in equation (4.16) is given by Mercer's theorem (Courant and Hilbert 1981; Cortes and Vapnik 1995): a kernel K($\mathbf{x}$, $\mathbf{y}$), with $\mathbf{x}, \mathbf{y} \in \Re^n$, is a dot product in some feature space, or $K(\mathbf{x}, \mathbf{y}) = \boldsymbol{\varphi}(\mathbf{x}) \cdot \boldsymbol{\varphi}(\mathbf{y})$, if and only if

$$K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x}) \quad \text{and} \quad \int\int K(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0, \ \forall f \in L^2.$$

Given such a kernel K, a possible set of functions $\boldsymbol{\varphi} = (\varphi_1, \varphi_2, \ldots)$ satisfying equation (4.16) can be determined from the eigenfunctions $\hat{\varphi}_i$ solution of the eigenvalue problem

$$\int K(\mathbf{x},\mathbf{y})\hat{\varphi}_i(\mathbf{x})d\mathbf{x} = \lambda_i\hat{\varphi}_i(\mathbf{y}), \tag{4.17}$$

with $\varphi_i = \sqrt{\lambda_i}\hat{\varphi}_i$. If the set of eigenfunctions $\hat{\boldsymbol{\varphi}}$ is finite, the kernel K is said to be *finite* and can be rewritten as

$$K(\mathbf{x},\mathbf{y}) = \sum \lambda_i\hat{\varphi}_i(\mathbf{x})\hat{\varphi}_i(\mathbf{y}), \tag{4.18}$$

where the sum ranges over the set of eigenfunctions. In the general case, the set $\boldsymbol{\varphi}$ is infinite, the kernel is said to be *infinite*, and the sum in equation (4.18) becomes a series or an integral.

4.2 SVM for regression

4.2.1 Regression estimation

We conclude this brief overview by discussing SVMs for regression estimation. In this case, the training set consists of N pairs $(\mathbf{x}_i, y_i)$ $(i = 1, 2, ..., N)$ with $\mathbf{x}_i \in \Re^n$ and $y_i \in \Re$. Similarly to SVMs for pattern recognition, in the case of linear regression SVMs look for the linear function $\mathbf{w}\cdot\mathbf{x}+b$ solution to the problem

$$\begin{array}{ll} \text{Minimize} & \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + C\sum(\xi_i+\xi_i^*) \\ \text{subject to} & y_i - \boldsymbol{w}\cdot\boldsymbol{x}_i - b \le \epsilon + \xi_i^* \\ & \boldsymbol{w}\cdot\boldsymbol{x}_i + b - y_i \le \epsilon + \xi_i \qquad i = 1, 2, \ldots, N \\ & \boldsymbol{\xi} \ge 0, \boldsymbol{\xi}^* \ge 0. \end{array}$$

The first two sets of constraints are derived from a function $|p|_\epsilon$, named *ϵ-insensitive loss function*, defined as

$$|p|_\epsilon = \begin{cases} 0 & \text{if } |p| < \epsilon \\ |p| - \epsilon & \text{otherwise} \end{cases}$$

with $p = y_i - \mathbf{w}\cdot\mathbf{x}_i - b$. Therefore, the term $\sum(\xi_i+\xi_i^*)$ is the ϵ-insensitive loss function evaluated over the training set for a given linear function. Note that for each i only one between ξ_i and ξ_i^* can be different form zero. In the general case of nonlinear regression and for a suitable kernel function K, the dual problem becomes

$$\begin{array}{ll} \text{Maximize} & -\frac{1}{2}(\boldsymbol{\alpha}^*-\boldsymbol{\alpha})^\top D(\boldsymbol{\alpha}^*-\boldsymbol{\alpha}) + \\ & \sum y_i(\alpha_i^*-\alpha_i) + \epsilon\sum(\alpha_i^*+\alpha_i) \\ \text{subject to} & \sum\alpha_i^* = \sum\alpha_i \\ & 0 \le \boldsymbol{\alpha}^* \le C, 0 \le \boldsymbol{\alpha} \le C. \end{array}$$

with $D_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ and α_i^* and α_i the Lagrangian multipliers associated to the variables ξ_i^* and

ξ_i respectively. The approximating function can then be expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) K(\mathbf{x}, \mathbf{x}_i) + b$$

where the only α_i^* or α_i strictly positive are those for which the constraints of the primal problem are satisfied with the equality sign. The corresponding $\mathbf{x}_i$ are again the support vectors, and the only points contributing to the sum for evaluating $f(\mathbf{x})$.

4.3 Kernels example

In this section we see some of the most used kernel, which have also the property of being easily implementable. Since in the *regression* case it is easier to show the different kernel characterisitcs, in the following examples we considered as a problem the regression of a set of points $\in R$.

4.3.1 Polynomial

$$K_p(\mathbf{u}, \mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^d - 1, \tag{4.19}$$

with $\mathbf{u}, \mathbf{v} \in [-a, a]^n$.
K_p is a symmetric function who satisfy the Mercer condition. Moreover, the mapping ϕ from input space X to feature space V (whose dimensionalty is $o(n^d)$) induced by K_p can be explicitly recovered from its definition (eq. 4.19). For example, for a polynomial of degree 2, given an input vector $\mathbf{x} = (x_1, \ldots x_n)$, the resulting mapping would be:

$$\phi(\mathbf{x}) = (x_1, \ldots x_n, (x_1)^2, \ldots (x_n)^2, x_1 x_2, \ldots x_n x_{n-1}). \tag{4.20}$$

This kernel has other significant properties. Particularly, it allows a fine tuning of the *VC-dimension* of the mapping functions. In fact, if we denote with R the radius of an hypersphere containing all the points projected in the feature space, then R depends on the degree d of the polynomial. As we have seen before, controlling the radius R means controlling also the *VC-Dimension* of the set of functions implemented by the kernel.

4.3.2 Gaussian Kernel

$$K(\mathbf{u}, \mathbf{v}) = exp(\frac{\|\mathbf{u} - \mathbf{v}\|^2}{2\sigma^2}), \sigma \in R. \tag{4.21}$$

This is an infinite kernel, and the resulting decision surface is a sum of gaussian functions, whose centers are the selected support vectors.

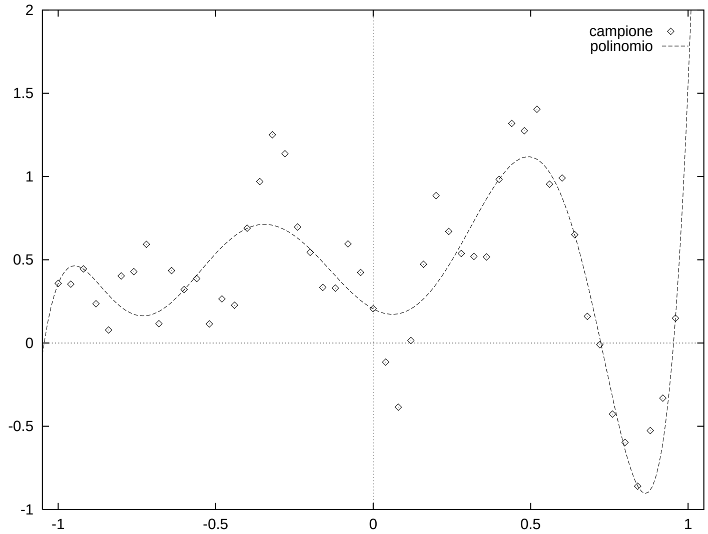

Figure 4.3: Curve approssimation using a polynomial kernel of degree 8 on a pseudo-random vector set.

4.3.3 Splines (one-dimensional)

$$K(u, v) = \sum_{p=0}^{n} u^p v^p + \sum_{i=1}^{N} (u - t_i)_+^n (v - t_i)_+^n, \tag{4.22}$$

Where n is the spline degree , $t_i \in R$ are the knots of the partition , and $(x)_+ = max\{x, 0\}$. This kind of kernel provides a linear functin in the $(n + N + 1)$-dimensional feature space generated by the following base functions

$$1, x, \ldots x^n, (x - t_1)_+^n, \ldots (x - t_N)_+^n,$$

the expression of the function in the input space R is:

$$f_n(x) = \sum_{p=0}^{n} a_p x^p + \sum_{i=1}^{N} b_i (x - t_i)_+^n.$$

As we can see in the figuree (4.5, 4.6), the shape of the resulting functions is often quite independent on the spline degree.

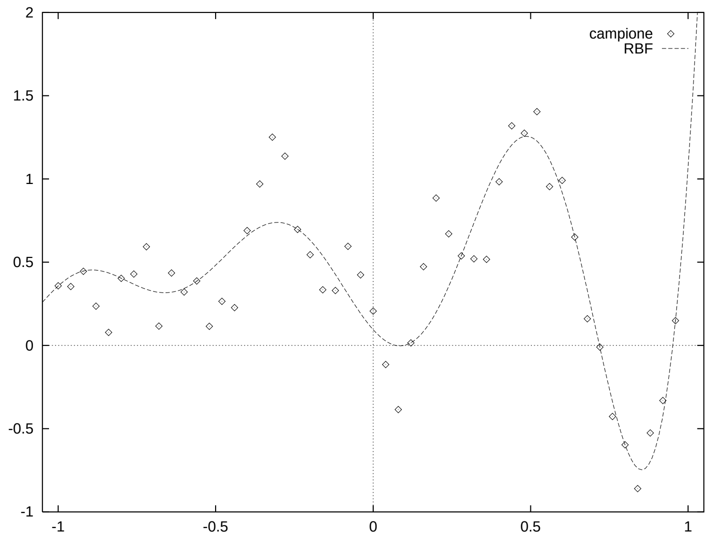

Figure 4.4: Example of a gaussian kernel. ($\sigma = 1.0$).

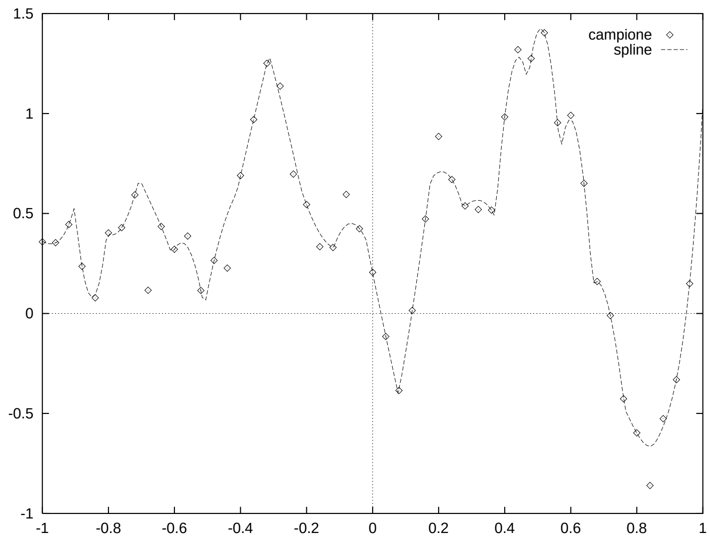

Figure 4.5: Spline kernel, degree=3, C=1000.

4.3.4 Fourier kernel

$$\begin{aligned} K(u,v) &= \tfrac{1}{2} + \sum_{n=1}^{N}[cos(nu)cos(nv) + sen(nu)sen(nv)) \\ &= \frac{sen[(N+1/2)(u-v)]}{sen\frac{u-v}{2}}, \end{aligned} \tag{4.23}$$

the $2N+1$-dimensional feature space related with this kernel is generated by

$$\frac{1}{\sqrt{2}}, cosx, senx, \ldots cosNx, senNx.$$

In practice, this kernel implements a Fourier transform. Given a function $f(x)$, its approximation ugin this kernel will be:

$$\frac{1}{\pi}\int_{-\pi}^{\pi} f(x-\tau)K(\tau,0)d\tau.$$

4.4 More about mapping into feature space

Let us spend a few more words about *mapping*. As we already pointed out, given a function $\phi : R^m \mapsto R^n, n > m$, we can use $< \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) >$ instead of $< \mathbf{x} \cdot \mathbf{y} >$. Moreover the Mercer theorem suggests which functions $K(\mathbf{x},\mathbf{y}) = (\phi(\mathbf{x}) \cdot \phi(\mathbf{y}))$ are *admissible* kernel and which are not (let us recall that the function $K(\mathbf{x})$ has to be a symmtric $L_2(R^n) \otimes L_2(R^n)$ function). Now we introduce a slightly different type of mapping.

4.4.1 Intermediate Mapping

In rough words, *intermediate mapping* is achieved by substituting the inner product in the feature space with an admissible kernel. This procedure ends with generating a new kernel. for example, if R is the input space, $\phi : R \mapsto R^m$ be the original mapping and $K_i \in L_2(R^m) \otimes L_2(R^m)$ be a kernel satisfying Mercer condition, then:

$$K(x,y) = K_i((\phi_1(x), \ldots \phi_m(x)), (\phi_1(y), \ldots \phi_l(y)))$$

also will satisfy the Mercer condition, and may be used as a new kernel.

4.4.2 Kernel tensor product

Let us consider again the spline kernel. If we choose the mapping $\psi_k(x) = (x - t_k)_+^p$, where t_k consitute a partition on f's dominion, then $K_i(x,y) = (\psi(x) \cdot \psi(y))$, that is

$$K(x,y)=\sum_{k=1}^{m}(x-t_k)_+^p(y-t_k)_+^p,$$

therefore

$$\begin{aligned} f(x) &= \textstyle\sum_{i=1}^{l}(\alpha_i^*-\alpha_i)K(x,x_i)+b \\ &= \textstyle\sum_{i=1}^{l}(\alpha_i^*-\alpha_i)\sum_{k=1}^{m}(x-t_k)_+^p(x_i-t_k)_+^p+b \\ &= \textstyle\sum_{k=1}^{m}(x-t_k)_+^p\left(\sum_{i=1}^{l}(\alpha_i^*-\alpha_i)(x_i-t_k)_+^p\right)+b \end{aligned}$$

As we can observe, the bigger is the input space dimension, the more are the mapping functions. To overcome this limitation we can use the tensor decomposition $R^n=\bigotimes_{i=1}^{n}R$, for decomposing a kernel defined on R^n in n factors, and each factor will be an admissible kernel for each input space dimension:

$$K(\mathbf{x},\mathbf{y})=\prod_{i=1}^{n}K_i(x_i,y_i).$$

Thus, the growth of the number of mapping function is linear, and this method allows us to easily construct spline kernels in more than one dimension.

4.4.3 Spline kernel (m-dimensional)

$$K(\mathbf{u},\mathbf{v})=K(u_1,v_1)K(u_2,v_2)\dots K(u_m,v_m). \tag{4.24}$$

Using a tensor product we can "assemble" a $(n+N+1)^m$-dimensional kernel with a sequence of one-dimensional spline kernels. If, for example, $m=2, n=1, N=2$, the base for the resulting space would be:

$$\begin{gathered} 1,x,y,xy, \\ (x-t_1)_+,(x-t_2)_+,(y-t_1)_+,(y-t_2)_+, \\ x(y-t_1)_+,x(y-t_2)_+,y(x-t_1)_+,y(x-t_2)_+, \\ (x-t_1)_+(y-t_1)_+,(x-t_1)_+(y-t_2)_+,(x-t_2)_+(y-t_1)_+,(x-t_2)_+(y-t_2)_+. \end{gathered}$$

4.4.4 Hilbert space mapping

Keeping our focus on spline kernels, let us note that the number and the structure of spline partitions are *delicate* parameters. Namely if we raise up te knot density, we would obtain both a greater flexibility of spline-generated fuctions, and an increased computational effort. A solution is represented by mapping from input space into an Hilbert space.
consider a mapping $R^n \mapsto \mathcal{H}$ such that

$$\begin{array}{rcl} \mathbf{x} & \longmapsto & \psi_{\mathbf{x}} \\ \Longrightarrow K(\mathbf{x},\mathbf{y}) & = & <\psi_{\mathbf{x}},\psi_{\mathbf{y}}>, \end{array}$$

where $<\cdot,\cdot>$ denotes the inner product of space $\mathcal{H}$. The obtained kernel satisfies (for construction) the Mercer conditions, since it is generated by the inner product of the feature space.
If we apply the former observation to the one-dimensional spline case we obtain:

$$\begin{array}{rcl} \psi_x(\xi) & = & (x-\xi)_+^p \\ \Longrightarrow K(x,y) & = & \int_t^T (x-\xi)_+^p (y-\xi)_+^p d\xi \\ & & \text{we suppose } t \le x \le y \le T \\ & = & \int_t^x (x-\xi)^p (y-\xi)^p d\xi \end{array}$$

$$\begin{array}{rl} & \text{substituting } \nu = (x-\xi) \text{ e } \delta = (y-x) \\ = & \int_0^{x-t} \sum_{j=0}^p \left(\begin{array}{c} p \\ j \end{array} \right) \nu^{p+j} \delta^{p-j} d\nu \\ = & \sum_{j=0}^p \frac{1}{p+j+1} \left(\begin{array}{c} p \\ j \end{array} \right) (x-t)^{p+j+1} (y-x)^{p-j} \end{array}$$

This is equivalent to the construction of a spline with an infinity-density knot parition, and moreover, its computation requires much less effort than the hight-but-finite -density case.

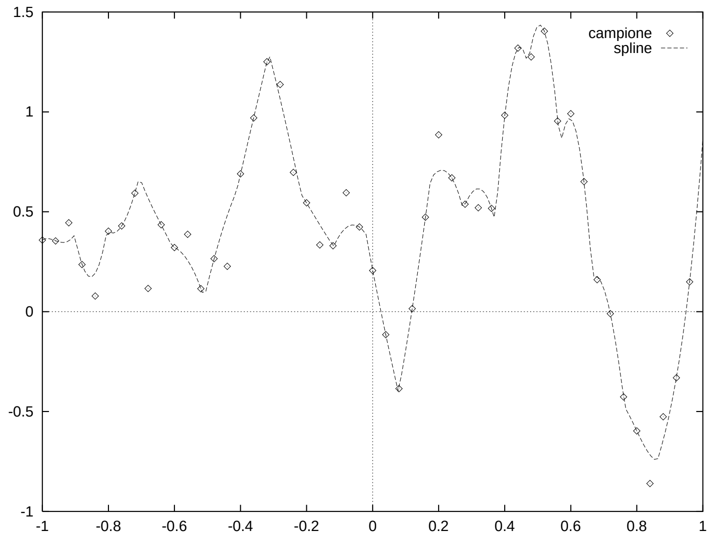

Figure 4.6: Spline kernel, degree=8, C=1000.

4.5 Implementation methods

In this section we present the algorithms for implementing SVM in the case of classification problems.

Let us start with the simple case in which the number of examples is not too large, say $l < 10^3$.In the following sections we will apprach also the case (more likely to happen) of very large *training sets.*

4.5.1 Training set of small size

In the previous section we have seen that the problem of training a SVM reduces to Problem **D2**. This is a QP problem with N box constraints and one equality constraint. The matrix D is symmetric, positive semi-definite and, in general, with a large number of nonzero elements.

When the training set is small, say less than a few hundreds of examples, it is possible to solve Problem **P4** by means of the *Complementary Pivoting Algorithm* (CPA) which we describe next.

4.5.1.1 Complementary Pivoting Algorithm

We first rewrite Problem **P4** in a more standard form and show that it can be transformed into a *Linear Complementary Problem* (*LCP*). Then we briefly review the *Complementary Pivoting Algorithm* (*CPA*), which solves *LCP*. For more details and a review of the main concepts and techniques of quadratic programming see (Bazaraa and Shetty, 1979).

4.5.1.1.1 Problem P4 revisited In what follows we let $l = l_1 + l_2$, $y_i = 1$ for $i = 1, 2, \ldots, l_1$, and $y_i = -1$ for $i = l_1 + 1, l_1 + 2, \ldots, l$. From equation (4.7) we have

$$\alpha_l = \sum_{i=1}^{l-1} \alpha_i y_i. \tag{4.25}$$

Substituting Equation (4.25) in the function of Problem **P4**, for the quadratic part we find

$$\boldsymbol{\alpha}^\top D\boldsymbol{\alpha} = \left(\sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i\right)^2 = \left(\sum_{i=1}^{l-1} \alpha_i y_i \mathbf{x}_i - \sum_{i=1}^{l-1} \alpha_i y_i \mathbf{x}_l\right)^2 = \sum_{i,j=1}^{l-1} \alpha_i H_{ij} \alpha_j$$

with $H_{ij} = y_i y_j (\mathbf{x}_i - \mathbf{x}_l) \cdot (\mathbf{x}_j - \mathbf{x}_l)$. Therefore, Problem **P4** becomes

Problem **P4'**

Minimize	$\frac{1}{2}\boldsymbol{\alpha}^\top H\boldsymbol{\alpha} - 2\sum \alpha_i$	
subject to	$0 \le \sum \alpha_i y_i \le C$	
	$0 \le \alpha_i \le C,$	$i = 1, 2, \ldots, l-1,$

with $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_{l-1})$ and the sums now ranging for $i = 1, 2, \ldots, l-1$. Problem **P4'** has the form of

Problem **P5**
Minimize $\quad \frac{1}{2}\boldsymbol{\alpha}^\top H\boldsymbol{\alpha} + \mathbf{k}\cdot\boldsymbol{\alpha}$
subject to $\quad A\boldsymbol{\alpha} \leq \mathbf{b}$
$\quad \boldsymbol{\alpha} \geq 0,$

with

$$\mathbf{k} = (\overbrace{-2, \ldots, -2}^{l_1}, \overbrace{0, \cdots, 0}^{l_2-1}), \qquad \mathbf{b} = (0, \overbrace{C, \cdots, C}^{l+1}),$$

and

$$A = \begin{pmatrix} -y_1 & \cdots & -y_{l-1} \\ y_1 & \cdots & y_{l-1} \\ & & \\ & I_{l-1} & \\ & & \end{pmatrix},$$

with I_{l-1} the $(l-1)\times(l-1)$ identity matrix.

4.5.1.1.2 Linear complementary problem In order to show that Problem **P5** can be transformed into a *LCP* we introduce the Lagrangian multipliers associated with the constraints. We thus let $\mathbf{u}$ and $\mathbf{v}$ be the $(l+1)$-D and $(l-1)$-D vectors of the Lagrangian multipliers associated with the constraints $A\boldsymbol{\alpha} \leq \mathbf{b}$ and $\boldsymbol{\alpha} \geq \mathbf{0}$ respectively. Then we denote with $\mathbf{s}$ the $(l+1)$-D vector of the *slack* variables[3] such that

$$s_i = b_i - \sum_{j=1}^{l-1} A_{ij}\alpha_j, \qquad i = 1, 2, \ldots, l+1.$$

The Lagrangian function can now be written as

$$L = \frac{1}{2}\boldsymbol{\alpha}^\top H\boldsymbol{\alpha} + \mathbf{k}\cdot\boldsymbol{\alpha} - \boldsymbol{\alpha}\cdot\mathbf{v} + \mathbf{s}\cdot\mathbf{u},$$

with $L = L(\boldsymbol{\alpha}, \mathbf{s}, \mathbf{u}, \mathbf{v})$. If we set the gradient of L with respect to $\boldsymbol{\alpha}$ equal to zero, we obtain

$$-H\boldsymbol{\alpha} - A^\top\mathbf{u} + \mathbf{v} = \mathbf{k}$$

[3]In the linear programming jargon, a slack variable is a nonnegative variable that turn an inequality into an equality constraint.

with the corresponding Kühn-Tucker conditions

$$\begin{aligned} \alpha_i v_i &= 0, \quad \text{for } i = 1, 2, \ldots, l-1 \\ s_i u_i &= 0, \quad \text{for } i = 1, 2, \ldots, l+1. \end{aligned}$$

Introducing the $2l$-D vectors of unknown $\mathbf{t} = (\mathbf{s}, \mathbf{v})$ and $\mathbf{z} = (\mathbf{u}, \boldsymbol{\alpha})$, Problem **P5** is equivalent to

Problem **P6**
Solve $\quad \mathbf{t} - M\mathbf{z} = \mathbf{q}$
subject to $\quad \mathbf{t}, \mathbf{z} \geq 0$
$\quad t_i z_i = 0, \qquad i = 1, 2, \ldots, 2l,$

with $\mathbf{q} = (\mathbf{b}, \mathbf{k})$, and

$$M = \begin{pmatrix} 0 & -A \\ A^\top & H \end{pmatrix}.$$

In the nonlinear programming literature, Problem **P6** is an example of *linear complementary problem.*

4.5.1.1.3 Complementary pivoting algorithm Let us first give an intuitive explanation of what *LCP* is about. Then we review the algorithm that we used to solve it.

Similarly to the case of linear programming, a solution to a *LCP* is a vertex of a polyhedral set. In the case of Problem **P6**, a solution vector $\mathbf{p} = (\mathbf{t}, \mathbf{z})$ is a vertex of the polyhedral set $S = \{\mathbf{p} : G\mathbf{p} = \mathbf{q}, \mathbf{p} \geq 0\}$, $G = [I_{2l}, -M]$. The attribute *complementary* is due to the constraint for which at least one between t_i and z_i, for all $i = 1, \ldots, 2l$, vanishes.

We now briefly summarize the main steps of an algorithm, named *Complementary Pivoting Algorithm* (*CPA*), able to solve Problem **P6**. This is the algorithm we have actually implemented for determining the *OSH* introduced in Section 2 and used to obtain the results illustrated in Section 4.

Observe that if $\mathbf{q} \geq 0$, then $\mathbf{t} = \mathbf{q}$ and $\mathbf{z} = 0$ is a solution. In our case, instead, $\mathbf{q}$ has many negative components. Thus, we introduce an *artificial* variable z_0 and consider the following modified problem

Problem **P6'**
Solve $\quad \mathbf{t} - M\mathbf{z} - \mathbf{1}z_0 = \mathbf{q}$
subject to $\quad \mathbf{t}, \mathbf{z} \geq 0, z_0 \geq 0$
$\quad t_i z_i = 0, \qquad i = 1, 2, \ldots, 2l.$

To find an initial vertex, we swap the artificial variable z_0 with the variable t_i corresponding to the most negative q_i. This vertex is not a solution of Problem **P6** because the artificial variable

z_0 is still different from zero. We thus need to explore the vertices of the Problem **P6'** until the variable z_0 vanishes. If we let $m = l + 1$ and $g_m = z_m$, the vertices are visited according to the following steps:

1. Let $\mathbf{d}_m$ be the column of the coefficients of the g_m variable. If $\mathbf{d}_m \geq 0$ go to step 4, else pivot at column $\mathbf{d}_m$. Let r be the pivot row. If this row contains z_0 go to step 3, else go to step 2.
2. The basic variable at row r is either t_l or z_l, for some $l \neq m$. Swap the variable g_m with the basic variable and let $g_m = t_l$ if the basic variable is z_l, or $g_m = z_l$ otherwise. Go to step 1.
3. Swap z_0 with g_m and exit (the current vertex is a solution of Problem **P6**).
4. Problem **P6** has no solution.

In our case, the fact that the matrix H has nonnegative elements with positive diagonal elements is sufficient to ensure the convergence of the algorithm (see Theorem 11.2.4 in (Bazaraa and Shetty, 1979)).

4.5.2 Training set of large size

Since the spatial complexity of CPA goes with the square of the number of examples, CPA cannot deal efficiently with more than a few hundreds of examples and in practice becomes useless with more than 1000 examples. Therefore, when the number of examples in the training set is large, like in many interesting computer vision applications, ad hoc procedures for training a large scale SVM must be studied.

In what follows we describe two different procedures useful to deal with training sets composed by a large number of examples, distinguishing between the case in which m, the dimension of the feature space, is low and the case of feature space of arbitrary, possibly infinite, dimension.

4.5.2.1 Chunking

In the case of feature space of finite dimension, the training of a SVM can be carried out without explicitly determining the support vectors, since the vector $\mathbf{w}$ can be obtained by solving directly Problem **P2** instead of the dual problem. Actually, if the dimension of the feature space is much smaller than the number of examples ($m \ll l$), this is also computationally more efficient.

In the more general case, the dimension of the feature space can be arbitrarily large, if not infinite. Then, we have necessarily to solve problem **D2**. Since the spatial complexity of CPA goes with the square of the number of examples and in many practical applications the size of the training set can be 10^5 or more, one has to resort to more sophisticated techniques.

In what follows we describe a method due to Osuna, Freud, and Girosi (1997) . In plain words the method can be summarized as follows. Exploiting the particular form of the objective function, the method finds the solution of the full problem **D2** by solving iteratively subproblems of **D2** which depend only on g variables (with g, typically, a small number). In each subproblem one maximizes the objective function of problem **D2** with respect to the selected g components of $\boldsymbol{\alpha}$ (the *chunk*), the other being kept constant. The obtained solution is then used to select a new set of g variables, with the components α_j violating the optimality conditions as possible candidates.

More precisely, we partition the set of index $I = \{1, \ldots, l\}$ in two sets: B, the *working set* and N, the *constant set*. Likewise, we decompose $\boldsymbol{\alpha}$ in the two vectors $\boldsymbol{\alpha}_B$ and $\boldsymbol{\alpha}_N$, and the matrix D in the four blocks:

$$D = \begin{pmatrix} D_{BB} & D_{BN} \\ D_{BN}^{\top} & D_{NN} \end{pmatrix}.$$

Then we consider the objective function of problem **D2** as a function of the $\boldsymbol{\alpha}_B$ only:

$$W(\boldsymbol{\alpha}_B|\boldsymbol{\alpha}_N) = -\frac{1}{2}\boldsymbol{\alpha}_B^T D_{BB}\boldsymbol{\alpha}_B + \boldsymbol{\alpha}_B \cdot \mathbf{h} + a,$$

with:

$$\mathbf{h} = \mathbf{1}_B - D_{BN}\boldsymbol{\alpha}_N,$$

and:

$$a = -\frac{1}{2}\boldsymbol{\alpha}_N^T D_{NN}\boldsymbol{\alpha}_N + \boldsymbol{\alpha}_N^T \cdot \mathbf{1}_N.$$

The subproblem for the working set B induced by the constant vector $\boldsymbol{\alpha}_N$ is:

Problem **R1**
Maximize $W(\boldsymbol{\alpha}_B|\boldsymbol{\alpha}_N)$
subject to $\sum_{i\in B} y_i\alpha_i = -\sum_{i\in N} y_i\alpha_i$
$0 \leq \alpha_i \leq C, \qquad i \in N$

The method consists of three steps:

1. Randomly select g index in I to form the set B and set $\boldsymbol{\alpha}_N = 0$.
2. Solve problem **R1** (for example by means of CPA).
3. Look for an index $j \in N$ such that one of the following is true:
    - $d(\mathbf{x}_j)y_j < 1$ and $\alpha_j = 0$
    - $d(\mathbf{x}_j)y_j = 1$ and $\alpha_j = 0$ or $\alpha_j = C$

- $d(\mathbf{x}_j)y_j > 1$ and $\alpha_j > 0$,

where $d(\mathbf{x}_j)$ is the signed distance of the point $\mathbf{x}_j$ from the current suboptimal separating hyperplane. If such an index j exists, then swap j with any index in B and go back to step 2. Otherwise stop: the vector $(\boldsymbol{\alpha}_B, \boldsymbol{\alpha}_N)$ is the solution of problem **D2**.

The proof of convergence for this method is given in (Osuna, Freund, and Girosi, 1997)[4]

In summary, depending on the dimension of the feature space we can choose between two different methods. If the dimension of the feature space is low, we solve directly the primal problem through LOQO (Vanderbei,1997), an algorithm implementing the interior point method which is available,for example, at http:://www.princeton.edu/rvdb. When, instead, the dimension of the feature space is large we can employ the method described above using CPA for solving each subproblem.

[4]The proof in (Osuna, Freund, and Girosi, 1997) is not rigorous and for pathological cases the theorem is wrong. This, however, has not been an issue in all practical applications.

Chapter 5

VIsual Dynamic Events REcognition

In this chapter we describe a system, called *VIDERE*, which is the first implementation of a reliable and versatile *machine*, able to learn the solution of a number of Computer Vision problems. We now move on introducing the system, which can be functionally divided into three modules: a *detection*, a *representation* and a *recognition* module.

5.1 Event detection

Since the *Dynamic Events* we are interested in take place in a frame sequence, the *detection* stage is basically a low-level image processing module dedicated to extract the relevant information from the image sequence. The *event detection* module may be separated into two logically different tasks: *raw acquisition* and *filtering*, though in practice these tasks are accomplished in parallel.

5.1.1 Raw Acquisition

This is the only module based on low-level visual processing. Since by hypothesis the camera is fixed and the background stationary, we employ a simple method which, while periodically updating the background image (kept fixed), starts off by locating all pixels of the current image at which the gray level difference with respect to the background image is larger than a fixed threshold m.

In fig 5.1 the low-level acquisition algorithm is sketched.The algorithm is very simple: each acquired image is compared with the current background image through a pixel-wise thresholded difference. The threshold can be either set by hand, or computed looking at the histogram of the image difference (typically the threshold is set to twice the standard deviation of the difference values). This generates a binary image where all the "white" pixels (later referred to just as *pixels*) are supposed to belong to the moving object. If the number of non-isolated pixels (isolated pixels are removed by a simple morphological operation) is sufficient (depending on the acquisition set-up and on the environment geometry) one is true among:

1. something has been detected, (wait = yes)
2. a new event is beginning (wait = no, flag = false)
3. an older event is being followed (flag = true)

In the first case we detect something but we want to check if it lasts for more than a few frames (as in highlights or specularity case). If after some frames the event is still active, we consider it as significant and process it. In the processing phase, the relevant information is extracted from the current image or just from the binary mask, and a set of features is computed. This low level processing depends on the particular application, and on the set of features we use to describe *framewise* the event. The algorithm is based on the hypothesis that the scenario can be described by a fixed background and a single moving object. Therefore during image acquisition we have that this condition is verified.

In fig. 5.2 the background refresh module is described.The background check (performed randomly) consists in acquiring a brief image sequence and putting it in a circular array containing the former background checks. A mean difference of the images from the acquired sequence is computed (for ensuring that nothing is currently moving) and, if the last images are consistent with the others no background refresh is done. Otherwise the last acquired image becomes the new background. If we changed the background while an event was being followed, we save the event as a *pseudo-event* (a further improvement of the system would take into account these "ambiguous" events).

5.1.2 Filtering

The output of the low-level acquisition module is a time series of raw data that represents the evolution of the parameter we chose for representing the event during the event course. We decided to filter with a simple Kalman procedure these data, in order to smooth the noise. The noise could depend on complex environment phenomenons like:

1. deformation of the extracted moving mask due to highlights and specularities.
2. Sharp variations of number of moving pixels due to interaction between the gray value of the moving objects and the color of the background pattern ,
3. or due to occlusions.

The first two problems can be solved using color images and a suitable morphological analysis of extracted moving mask, but we decided not to apply any kind of *apriori* knowledge about the problem, in order to make it the most flexible and *ad hoc*. Thus we used a Kalman procedure to filter the raw data. In the following we briefly introduce the Kalman filtering procedure.

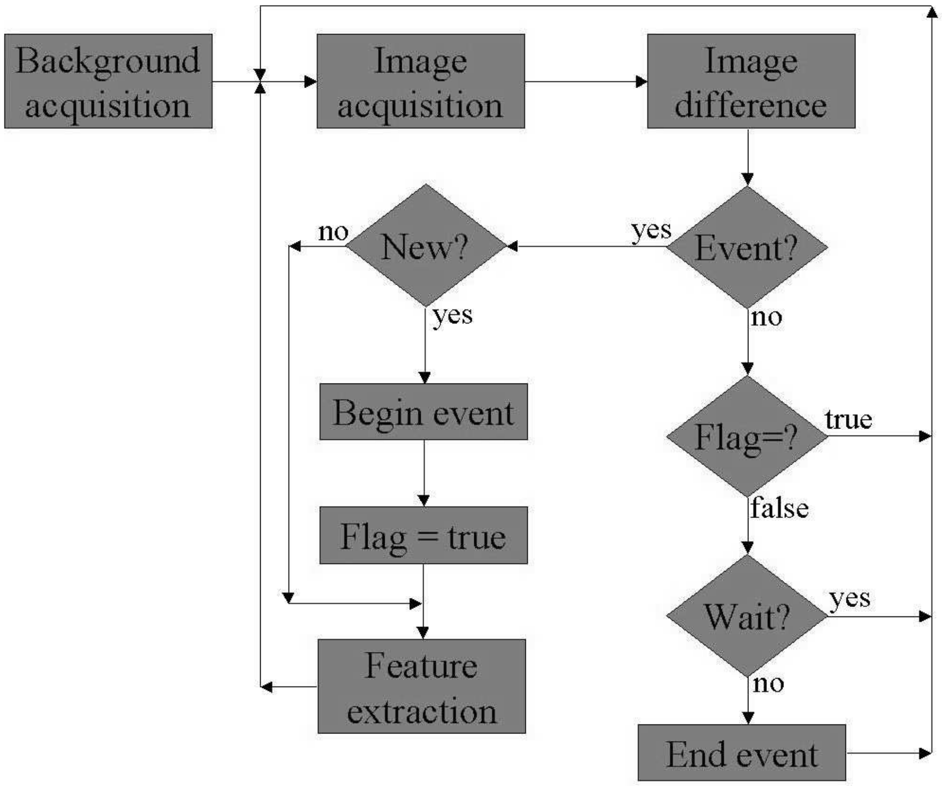

Figure 5.1: Scheme of *image acquisition and event management* module

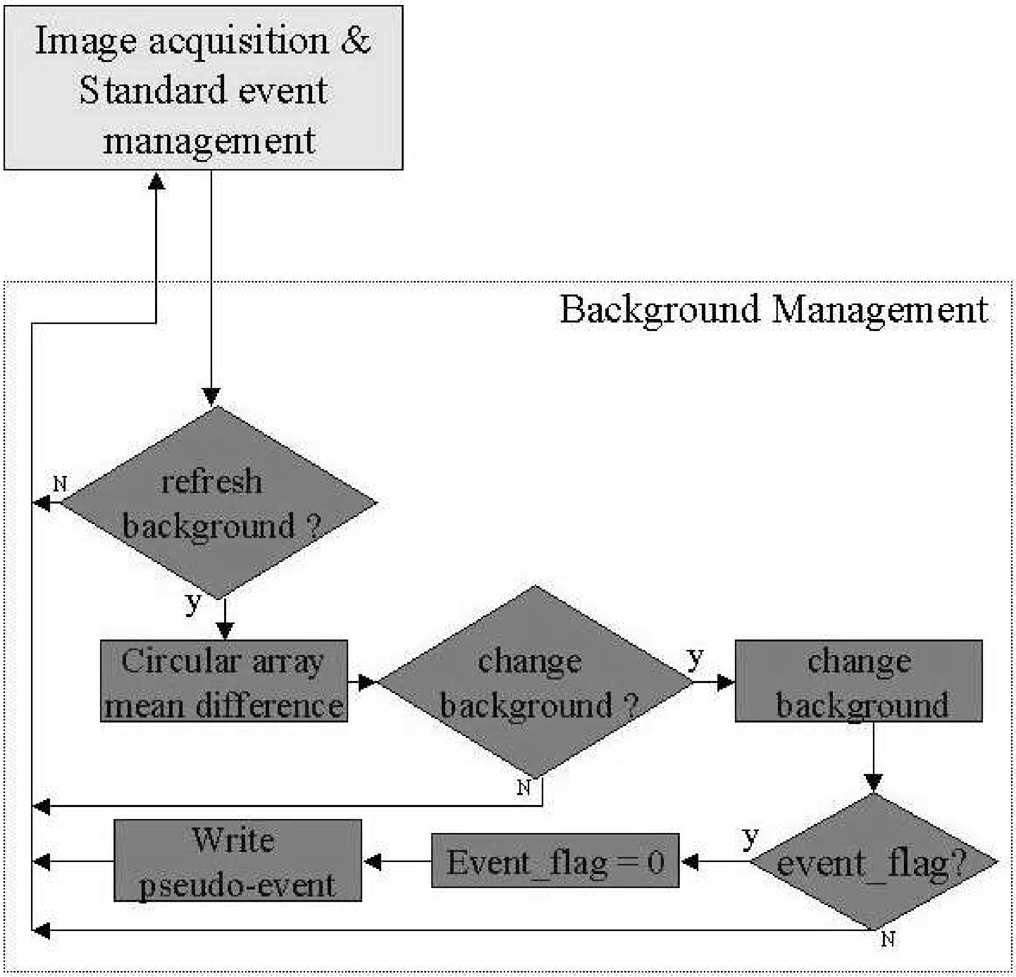

Figure 5.2: Scheme of the *background management* module

5.1.2.1 Kalman Filtering Technique

Kalman filtering, is a general method for integrating noisy measurements.

The behavior of a dynamic system can be described by the evolution of a set of variables, called state variables. In practice, the individual state variables of a dynamic system cannot be determined exactly by direct measurements; instead, we usually find that the measurements that we make are functions of the state variables and that these measurements are corrupted by random noise. The system itself may also be subject to random disturbances. It is then required to estimate the state variables from the noisy observations.

If we denote the state vector by $\mathbf{s}$ and denote the measurement vector by $\mathbf{x}'$, a dynamic system (in discrete-time form) can be described by

$$\begin{aligned} s_{i+1} &= H_i(s_i) + n_i, \ i = 0, 1, \ldots, N \\ \phi_i(\mathbf{x}'_i, s_i) &= 0, \ i = 0, 1, \ldots, N \end{aligned} \tag{5.1}$$

where the 5.1 denotes the relationship between the system state and the measurement vector $\mathbf{x}'_i$, and n_i is the vector of random disturbance of the dynamic system and is usually modeled as white noise:

$$E[n_i] = 0 \quad and \quad E[n_i {n_i}^T] = Q_i$$

In practice, the system noise covariance Q_i is usually determined on the basis of experience and intuition (i.e., it is guessed). In practice, the measurements that can be made contain random errors. We assume the measurement system is disturbed by additive white noise, i.e., the real observed measurement $\mathbf{x}_i$ is expressed as

$$\mathbf{x}_i = \mathbf{x}'_i + \eta_i,$$

where

$$E[\eta_i] = 0$$

$$E[\eta_i {\eta_j}^T] = \begin{cases} \Lambda_i & \text{for} \quad \text{i=j}, \\ 0, & \text{for} \quad i \neq j \end{cases}$$

The measurement noise covariance Λ_i is either provided by some signal processing algorithm or guessed in the same manner as the system noise. In general, these noise levels are determined independently. We assume then there is no correlation between the noise process of the system and that of the observation, that is

$$E[\eta_i {\eta_j}^T] = 0$$

for every i and j.

Linear Kalman Filter

When $H_i(s_i)$ is a linear function

$$s_{i+1} = H_i s_i + n_i$$

and we are able to write down explicitly a linear relationship

$$\mathbf{x}_i = F_i s_i + \eta_i$$

from $f_i(\mathbf{x'}_i, s_i) = 0$, then the standard Kalman filter is directly applicable.

- State Prediction:
$$\hat{s}_{i|i-1} = H_i \hat{s}_{i-1}$$

- State covariance matrix prediction:
$$P_{i|i-1} = H_{i-1} P_{i-1} H_{i-1}^T + Q_{i-1}$$

- Kalman gain matrix:
$$K_i = P_{i|i-1} F_i^T (F_i P_{i|i-1} F_i^T + \Lambda_i)^{-1}$$

- State estimation update:
$$\hat{s}_i = \hat{s}_{i|i-1} + K_i (x_i - F_i \hat{s}_{i|i-1})$$

- State covariance matrix update:
$$P_i = (\mathbf{I} - K_i F_i) P_{i|i-1}$$

- Initialization:
$$P_{0|0} = \Lambda_{S_0}$$
$$\hat{s}_{0|0} = E[s_0]$$

Figure 5.3 is a block diagram for the Kalman filter. At time t_i , the system model inherently in the filter structure generates $\hat{s}_{i|i-1}$, the best prediction of the state, using the previous state estimate $\hat{s}_{i-1}$. The previous state covariance matrix is extrapolated to the predicted state covariance matrix $P_{i|i-1}$. $P_{i|i-1}$ is then used to compute the Kalman gain matrix K_i and to update the covariance matrix P_i . The system model generates also $F_i \hat{s}_{i|i-1}$ which is the best prediction of what the measurement at time t_i will be. The real measurement x_i is then read in, and the measurement residual (also called innovation)

$$r_i = x_i - F_i \hat{s}_{i|i-1}$$

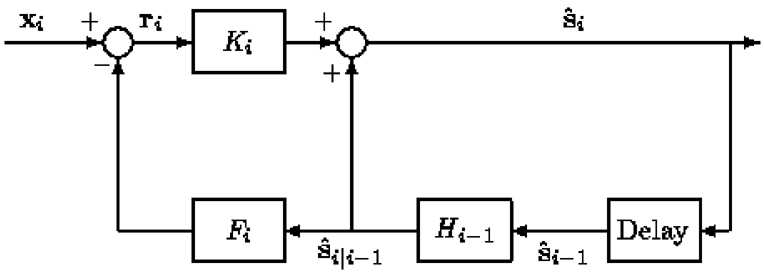

Figure 5.3: Kalman filter block diagram

is computed. Finally, the residual r_i is weighted by the Kalman gain matrix K_i to generate a correction term and is added to $\hat{s}_{i|i-1}$ to obtain the updated state $\hat{s}_i$.

The Kalman filter gives a linear, unbiased, and minimum error variance recursive algorithm to optimally estimate the unknown state of a linear dynamic system from noisy data taken at discrete real-time intervals. Without entering into the theoretical justification of the Kalman filter, for which the reader is referred to many existing books such as [FZ92], we insist here on the point that the Kalman filter yields at t_i an optimal estimate of s_i , optimal in the sense that the spread of the estimate-error probability density is minimized. In other words, the estimate $\hat{s}_i$ given by the Kalman filter minimizes the following cost function

$$\mathcal{F}_i(\hat{s}_i) = E[(\hat{s}_i - s_i)^T M(\hat{s}_i - s_i)],$$

where M is an arbitrary, positive semidefinite matrix. The optimal estimate $\hat{s}_i$ of the state vector s_i is easily understood to be a least-squares estimate of s_i with the properties that [4]:

1. the transformation that yields $\hat{s}_i$ from $[x_0^T \cdots x_i^T]^T$ is linear,
2. $\hat{s}_i$ is unbiased in the sense that $E[\hat{s}_i] = E[s_i]$,
3. it yields a minimum variance estimate with the inverse of covariance matrix of measurement as the optimal weight.

By inspecting the Kalman filter equations, the behavior of the filter agrees with our intuition. First, let us look at the Kalman gain K_i. After some matrix manipulation, we express the gain matrix in the form:

$$K_i = P_i F_i^T \Lambda_i^{-1} \tag{5.2}$$

Thus, the gain matrix is "proportional" to the uncertainty in the estimate and "inversely proportional" to that in the measurement. If the measurement is very uncertain and the state estimate is relatively precise, then the residual r_i is resulted mainly by the noise and little change in the state estimate should be made. On the other hand, if the uncertainty in the measurement is small and that in the state estimate is big, then the residual r_i contains considerable information about errors in the state estimate and strong correction should be made to the state estimate. All these are exactly reflected in 5.2.

Now, let us examine the covariance matrix P_i of the state estimate. By inverting P_i and replacing K_i by its explicit form 5.2, we obtain:

$$P_i^{-1} = P_{i|i-1}^{-1} + F_i^T \Lambda_i^{-1} F_i \tag{5.3}$$

From this equation, we observe that if a measurement is very uncertain (Λ_i is big), the covariance matrix P_i will decrease only a little if this measurement is used. That is, the measurement contributes little to reducing the estimation error. On the other hand, if a measurement is very precise (Λ_i is small), the covariance will decrease considerably. This is logic. As described in the previous paragraph, such measurement contributes considerably to reducing the estimation error. Note that Equation 5.3 should not be used when measurements are noise free because Λ_i^{-1} is not defined.

A brief note about Kalman Filtering

The Kalman filter formalism works under the assumption that the system and the measurement noises are uncorrelated and that they are all Gaussian white (for additional informations see [COOP86].
There exist many other methods to solve the parameter estimation problem: general minimization procedures, weighted least-squares method, and the Bayesian decision-theoretic approach. However we chose the Kalman filter approach as our main tool to solve the parameter estimation problem, because the Kalman Filter

- gives a linear, unbiased, and minimum error variance algorithm,
- takes measurements into account incrementally (recursivity), and
- is a simple and efficient procedure to solve the problem (computational tractability).

5.2 Event representation

The choice of the representation to adopt is a crucial aspect of the entire system. The fact that the event to be represented neither lasts the same amount of time, nor has the same spatio-temporal evolution (not even within events corresponding to the same class) makes this choice particularly difficult. For the time being, we adopted a very simple representation that satisfies the basic requirement of providing a stereotyped representation.

Let us suppose that at the end of the detection phase the event i of a class be descripted by a set of M T_i-dimensional vectors (where T_i is the event duration in number of frames, and M is the number of different features we chose)

$$F_1^i = F_1[1], \ldots, F_1[T_i],$$
$$\ldots$$
$$F_M^i = F_M[1], \ldots, F_M[T_i]$$

where $F_j[k] \in \Re$ are called *raw feature vectors*. Each class $C_j,\ j = 1, \ldots, N_{class}$ will be represented at this point by a set

$$C_j = \{F_1^i, \ldots, F_M^i \mid i = 1, \ldots, N_j\} \quad , \qquad j = 1, \ldots, N_{class}$$

Whilst the number M of features is kept fixed in a specific application, each istance of an event type can last for a different number T of frames. If, for example, we think of a person walking from a room to another room, we can easily observe that the same action can be performed in many different ways and times. Therefore we need a *stereotyped* representation, able to take into account these characteristics in a single standard, event-independent structure. In practice, using the raw data extracted from the detection module, we construct a *feature vector* with a fixed number of components for each event and with each component carrying comparable information across all events. To construct such a *feature vector* for each event i we create a $M \times N$-dimensional vector $\mathbf{x}$, where N is the number of components with which we want to describe each *raw feature vector*. The first N components of vector $\mathbf{x}$ will be extracted from the *raw feature vector* F_1^1, the following N from the vector F_2^i, and so on. The resulting vector carries on informations about all the raw feature vectors, and *has a fixed dimension*, that is indipendent from the particular event or class.

The problem that arises is to find out an effective method to represent the *raw feature vectors* with a number N of parameters without losing information. the simplest method is to approximate each raw feature vector with a suitable parametric function, and take the resulting parameters as a description of the raw vector itself. For example, if we suppose that each raw vector can be approximated by a straight line we will have $N = 2$ components, leading to a $M \times 2$-dimensional *feature vector* for each event. If we would choose a polynomial representation of, say, degree 5 we will have $M \times 5$-dimensional *feature vectors*.

5.2.1 Representation using *sampling*

The simplest method to extract a fixed lenght representation from the *raw feature vectors* is the direct sampling of the vector itself. In plain words, we can choose a sample number N_s such that $N_s \leq T_i \quad \forall i = 1, \ldots, N_{class}$ and pick up N_s equally distributed values $\hat{F}_{j,1}^i, \ldots, \hat{F}_{j,N_s}^i$ from each raw vector F_j^i, composing a new $M \times N_s$-dimensional feature vector. The problem of this method is that we do not have *a priori* informations about the distribution of the lenghts T_i of the events, then it is very difficult to guess a suitable sample value N_s (if it is too small we cannot ensure a proper description).

A better approach consists in approximating each raw feature vector by mean of a suitable class of functions, then sampling each approximated function as described above. In this way we can choose any number N_s of samples, independently of the particular event duration T_i.

A variation of this method has been proposed for the current system implementation. Our method consists in approximating each *raw feature vector* with a parametric function, and using the computed parameters as a description of the vector. This approach has several vantages, namely:

- the dimension of the resulting vector is lower than in the sampling case, without compromising the description quality,
- this description is less noise-dependent,
- less parameters are to "guessed".

Among the many possible classes of parametric approximating functions, we chosethe spline functions.

5.2.2 Representation using *splines*

Spline functions are piecewise polynomial functions suitable for a compact and flexible function approximation. In order to accomplish this task we decided to use SVM's for regression. The reasons for these choices are

- splines provide a easy-to-tune *raw feature vector* representation
- the SVM's spline implementation is easy (that is there is a suitable Kernel function)
- the computational complexity of the solution depends on the number of support vectors that one needs to approximate the raw vector with ϵ-accuracy, rather than the dimensionality of the space or the number of knots

Let us describe the kernel for the approximation of the one-dimensional *raw feature vector* F_h^k on the interval $[0, T_k]$ by splines of order $d \geq 0$ with m knots (see [**?**], [**?**]).

$$t_1, \ldots, t_m, \qquad t_i = \frac{iT_k}{m}, \qquad i = 1, \ldots, m$$

The spline approximation has the following form (see fig. 5.4):

$$F_h^k(x) = \sum_{r=0}^{d} \hat{a}_r x^r + \sum_{j=1}^{m} a_j (x - t_j)_+^d \qquad (5.4)$$

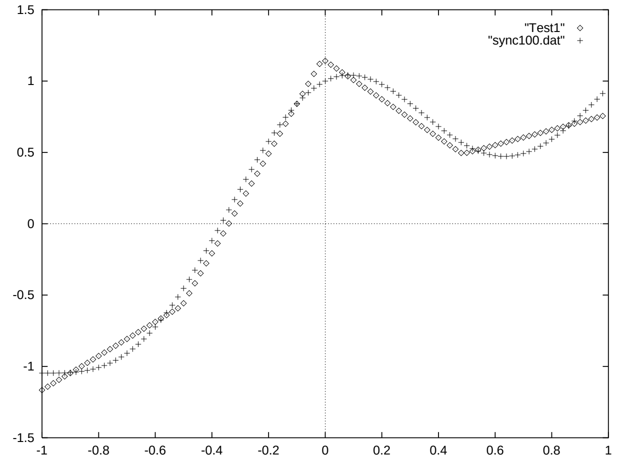

Figure 5.4: Example of a 4 partitions linear spline regression.

Now consider the mapping of the one-dimensional variable x into the $(m + d + 1)$-dimensional vector u:

$$x \longrightarrow u = (1, x, \ldots, x^d, (x - t_1)_+^d, \ldots, (x - t_m)_+^d)$$

where

$$(x - t_i)_+^d = \begin{cases} 0 & \text{if } x \leq t_i \\ (x - t_i)^d & \text{if } x > t_i \end{cases}$$

We can thus observe that spline approximation can be viewed as an inner product between the two vectors

$$F_h^k(x) = (a \cdot u) \qquad (5.5)$$

where $a = (a_0, \ldots, a_{m+d})$. Therefore we can construct a kernel that generates the inner product in the *raw vector feature* space as follows:

$$K(x, x_t) = \sum_{r=0}^{d} x_r x_t^r + \sum_{i=1}^{m} (x - t_i)_+^d (x_t - t_i)_+^d \tag{5.6}$$

By using this kernel, the SV Machine constructs the function

$$F_h^k(x, \beta) = \sum_{i=1}^{l} \beta_i K(x, x_i) + b \tag{5.7}$$

That is, we obtain a spline of order d defined on m knots and constructed by using the l support vectors (in our case $l \leq T_k$). Now, if we substitute eq 5.6 into 5.7 we obtain

$$F_h^k(x, \beta) = b + \sum_{r=0}^{d} x^r (\sum_{i=1}^{l} x_i^r) + \sum_{j=1}^{m} (x - t_j)_+^d (\sum_{i=1}^{l} \beta_i (x_i - t_j)_+^d)$$

If we compare the last equation with the 5.4 we can again define the resulting *raw vector approximation* as an inner product

$$F_h^k(x, \beta) = (a \cdot u)$$

where

$$a = (b + \sum_{i=1}^{l} \beta_i, \sum_{i=1}^{l} \beta_i x_i, \ldots, \sum_{i=1}^{l} \beta_i x_i^d, \sum_{i=1}^{l} \beta_i (x_i - t_1)_+^d, \ldots, \sum_{i=1}^{l} \beta_i (x_i - t_m)_+^d)$$

is a $(m + d + 1)$-dimensional parameter vector based on the computed support vectors. Thus, we can use this parameter vector as a synthetic representation of the single raw feature vector. Now in order to have the representation of the entire event we can just compose a $(M \times (m + d + 1))$-dimensional feature vector using the single parametric representations seen above.

5.3 Event recognition

This last module of *VIDERE* is only used in the test stage. The multiclass classification is a difficult problem, and an optimal solution is yet to be found. Notwithstanding, several methods has been proposed to provide a practical effective solution. Since we have a good two-class classifier, (Support Vector Machine), we adopted two simple methods that perform multiclass classification using a combination of two-class decision rules. Let us suppose that for each couple of classes C_i C_j , a training stage provides us with a SV Machine which classifies an unknown (event) vector $\mathbf{x}$ according to the linear rule (see chapter 4)

$$f_{i,j}(\mathbf{x}) = sign(\sum_{i=h}^{l} \alpha_h y_h K(\mathbf{x}, \mathbf{x}_h)) \tag{5.8}$$

where the kernel K determines the non-linear decision surface in the input space.

One vs. All Let us suppose that we have 4 different labels. Then we can construct a set of 8 different classes composed by the original 4 labels plus 4 classes made by all the labels but one. Namely, if a number ranging from 1 to 4 represents a label, we would have the classes:

$$\{1\}\ ,\ \{2\}\ ,\ \{3\}\ ,\ \{4\}\ ,\ \{2,3,4\}\ ,\ \{1,3,4\}\ ,\ \{1,2,3\}$$

Now we can compute 4 decision rules like 5.8 (the first between $\{1\}$ and $\{2,3,4\}$, the second between $\{2\}$ and $\{1,3,4\}$, and so on). The unknown vector $\mathbf{x}$ is classified according to these rules, an from the classification results we could argue the original label of the unknown vector. For instance, if the vector belongs to label 3 we would obtain

$$-1\ ,\ -1\ ,\ 1\ ,\ -1$$

Tennis Tournement The classification of a new event $\mathbf{x}$ is obtained by means of a linear SVM trained to distinguish between all possible pairs of different event classes. Let us suppose that we have 8 different classes. At the end of the training we have $\frac{n(n-1)}{2}$ rules like the 5.8 (in our case $\frac{8 \cdot 7}{2} = 28$ different hyperplanes).

As proposed in Pontil and Verri [?] the multiclass classification follows the rules of a *tennis tournament* (see Figure 5.5 for an example). Each class (or label) is a *player* and at each match the winner is determined by the label temporarily assigned to the unknown event $\mathbf{x}$. The final determines the label assigned to the event.

Note 2 *This method is theoretically* order-dependent. *That is, its results could depend on the order which we follow while playing the matches. In spite of this apparent bound, as we will point out later, our experiments confirmed its effectiveness and relability over the range of applications chosen to test the system.*

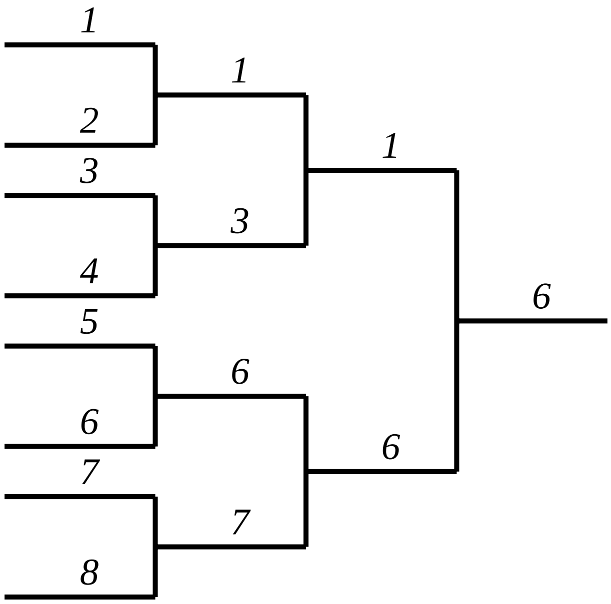

Figure 5.5: An unknown event $\mathbf{x}$ is assigned the label "6" following the rules of a tennis tournament (see text).

Chapter 6

Applications

6.1 People monitoring

We considered the task of *detecting* and *classifying* dynamic events from a sequence of images acquired by a camera looking down a corridor from a fixed position. In fig 6.1 we can see the scenario and some event trajectories are drawn. Let us note that despite the *triviality* of considered events (one might argue that it suffices to find the door's location) the choice of discarding any *a priori* geometric information and the presence of a flat perspective on the scene makes the visual task more difficult. Occasionally, one or more people are expected to walk up or down the corridor, possibly entering or leaving one of the adjacent rooms. The goal is to build a system able to learn from examples to

- detect the presence of moving people in the observed scene, and
- recognize their trajectory.

6.1.1 Raw acquisition

The low-level acquisition phase follows the scheme depicted in fig. 5.1. If a *significant* change is detected we

1. form a minimal rectangular box of center $\mathbf{c}$ enclosing all nonisolated pixels where the change has been detected;
2. provide the filtering module with these informations, in order to track the box over the image sequence until the change remains significant.

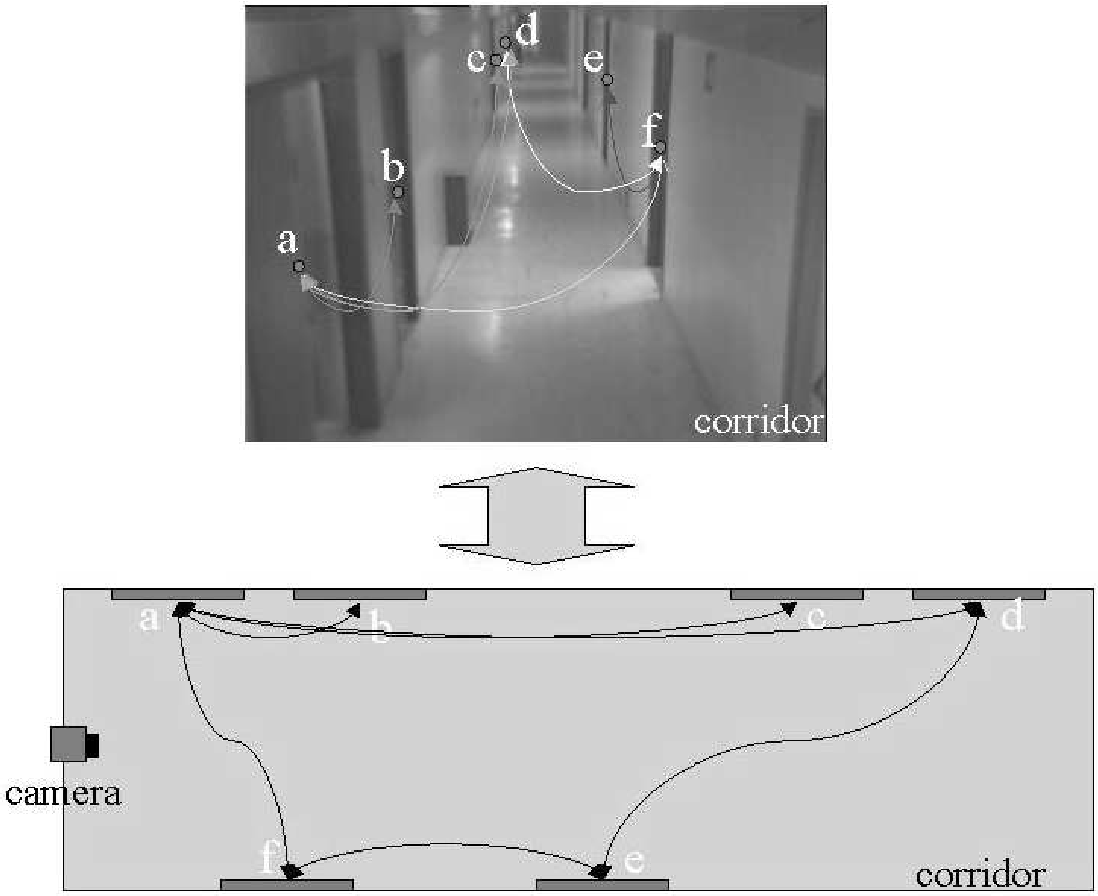

Figure 6.1: **Top**: actual view of the camera. **Bottom**: *Bird* view of the corridor and some events (note that this is not the vantage point of the camera).

Figure 6.2: Typical frame from a sequence containing an event.

A change is considered *significant* if the number N of connected pixels over which a change is detected exceeds a given threshold t_1, and *remains* significant until N remains higher than a second (lower) threshold t_2. Events that last for less than a minimum number of frames, f, are discarded as noise. An example of this module at work is illustrated in Figures 6.2, 6.3, 6.4, 6.5.

This module depends on the four thresholds m, t_1, t_2, and f. While t_1, t_2, and f were set manually observing a number of sequences beforehand, the threshold m is set to twice the standard deviation σ of the histogram of the pointwise signed image difference between two frames (both acquired while the camera looks at the stationary background).

the simple procedure depicted in fig. 5.2 and described in the last chapter has been employed to deal with a background that could be changing with time (some of the corridor lights switched on or off, or a door left open or closed). The reasons for using such a simple event acquisition and background management procedure are:

1. the light conditions in the considered environment are very stable and time-independent (no windows, neither light switches in the corridor)
2. the absence of any independently moving object (as courtains, plants, light furnitures as chairs)

Figure 6.3: A change in the image sequence has been detected. The white points are those considered as *moving* with respect to the background.

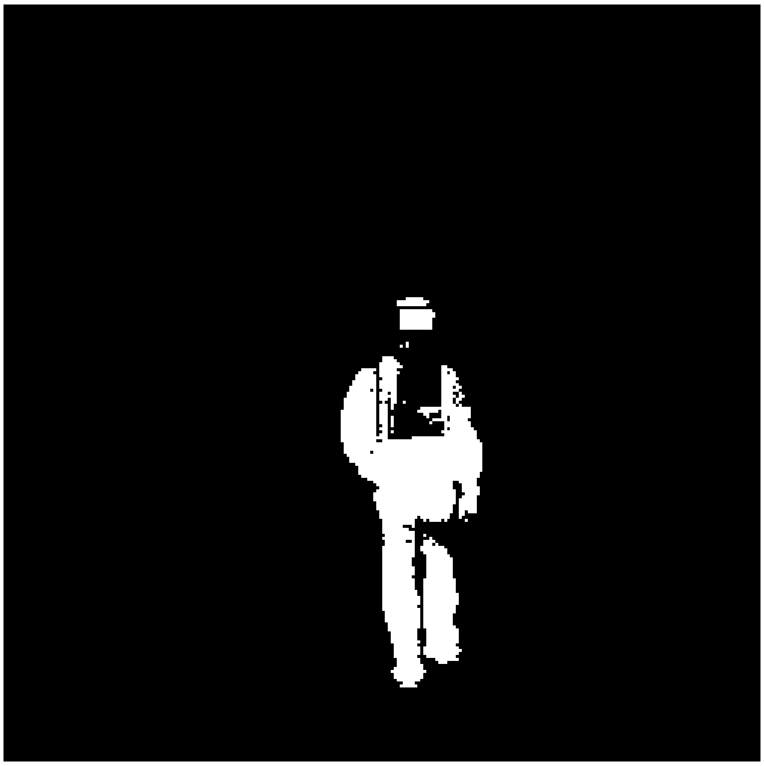

Figure 6.4: The relevant information is extracted from the current image.

Figure 6.5: The minimal box encloses all the non-isolated *moving* points.

As we can see from figures 6.3 6.4 the results of the low-level module are rather good, and their *light* computability makes them efficient also in a real-time on-line system.

Note 3 *The system is able to detect and recognize* one *dynamic event at a time. The realization of a suitable* data association *procedure and an improved filtering section for dealing with multiple moving people was beyond the aim of this thesis, but is being currently implemented in a new real-time version of the VIDERE system.*

6.1.2 Filtering

In this specific application we chose as *event-describing variables* the (2-dim) coordinates of the center of the enclosing box and the number of non-isolated pixels cantained in the box. This means that, if we suppose that the event lasts for T_N frames, than we obtain a T_N-long 3-dim vector defined in the input space $[0, image\_width] \times [0, image\_height] \times \Re$. We assumed these three parameters as independent, therefore we used three independent one-dimensional Kalman filters, where the evolution of the system is given by:

$$s_i = H_i s_{i-1} + n_i$$

where $s_i = [f_i, v_{f,i}]^T$ is the state of the system at time i, f is the feature to track, v_f is its velocity and the evolution matrix is simply:

$$H_i = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and $n_i = [\eta_i, \eta_{v,i}]^T$ is the additive noise.

6.1.3 Event representation

The representation is built in three steps. In the first step, the detected event is described by three T-dimensional vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{a}$ (where T is the event duration in number of frames). The vectors $\mathbf{u} = (u_1, u_2, \ldots, u_T)$ and $\mathbf{v} = (v_1, v_2, \ldots, v_T)$ consist of the horizontal and vertical positions on the image plane of the box center $\mathbf{c}$, while the vector $\mathbf{a} = (a_1, a_2, \ldots, a_T)$ of the number of pixels N within the box at each frame.

In the second step, the vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{a}$ are normalized between $[0, 1]$ in both range and time and regarded as the sampling of three functions

$$\begin{aligned} U &: [0,1] \to [0,1] \\ V &: [0,1] \to [0,1] \\ A &: [0,1] \to [0,1]. \end{aligned}$$

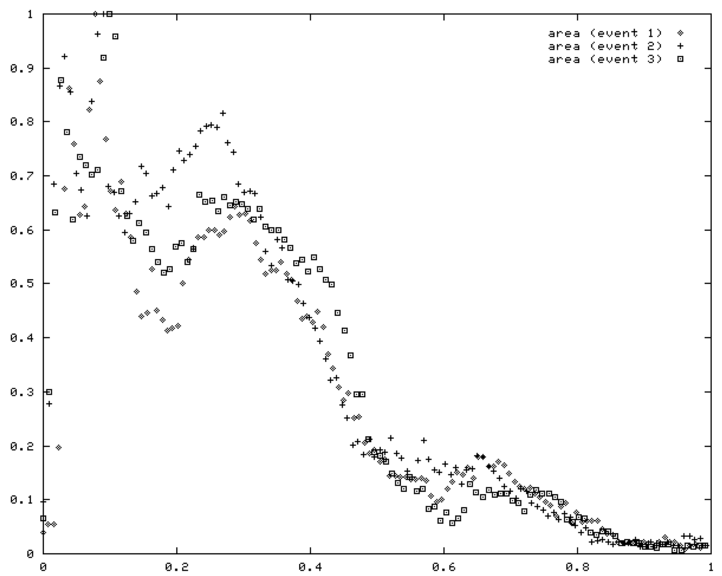

Figure 6.6: Normalized number of pixels in the enclosing box

In fig. 6.6,6.7,6.8 we see the normalized features as a function of time. Each plot contains three itances of the same event (that is three time series of the same feature extracted from three events belonging to the same class).

In the third and final step an SVM for regression is run on the data **u**, **v**, and **a** to build a representation consisting of an array of numbers of fixed size and homogeneous meaning across the events. This is obtained through the use of a kernel K corresponding to a piecewise linear spline with ℓ equally spaced fixed knots in the unit interval (Vapnik 1998) (with $\ell = 10$ in the present implementation). This means 12 spline coefficients for each vector. Let us denote with

$$u^{sp}[i], v^{sp}[i], a^{sp}[i] \qquad i = 1, \ldots, 12$$

the resulting spline vectors. In order to both reduce further on the final dimension of feature vectors, and to perform an ultimate smoothing, we convolved the 12 spline coefficients with the one-dimensional low-pass mask $m = [1, 2, 4, 2, 1]$ obtaining a new reduced spline vector. For u^{sp} for example, we obtain:

$$\hat{u}^{sp}[i] = \frac{1}{10} \sum_{j=i-3}^{10} m[j] u^{sp}[j], \quad i = 2, \ldots, 10$$

The detected event is finally described by means of a 25-dimensional vector $\mathbf{x}$ the components of which are obtained as follows. The first 8 components are the spline coefficients obtained from the regression estimation on the vector **u**, the second 8 from **v**, and the third 8 from **a**. In the attempt to make use of the information contained in the duration of the observed action (effectively lost in the previous normalization stages), the last component is the elapsed time T

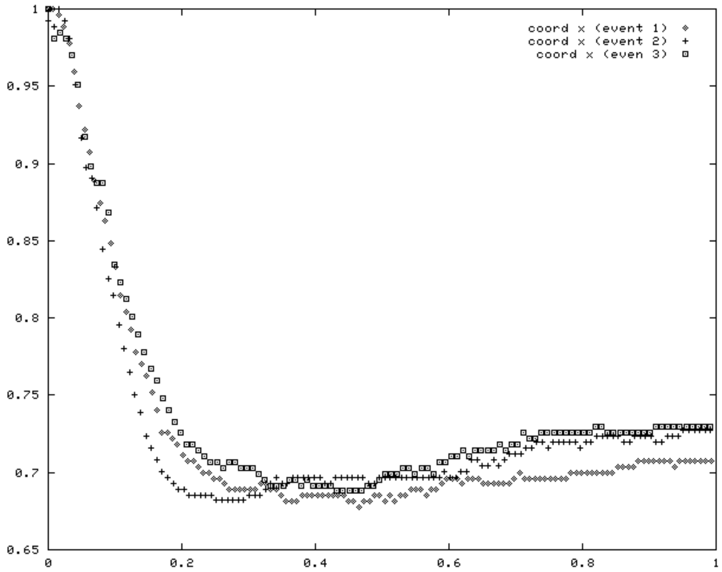

Figure 6.7: Normalized x coordinate of enclosing box

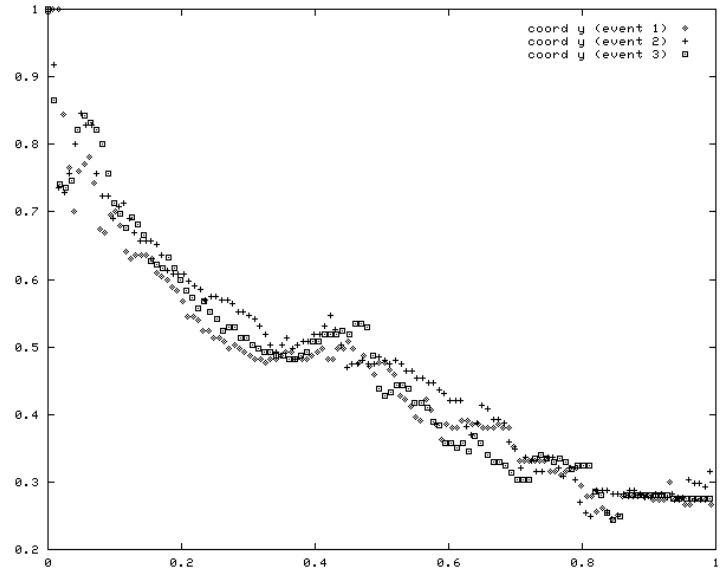

Figure 6.8: Normalized y coordinate of enclosing box

of the event. In Fig. 6.9 are reported the three feature vectors obtained from the events whose raw components are showed in Fig. 6.6,6.7,6.8.

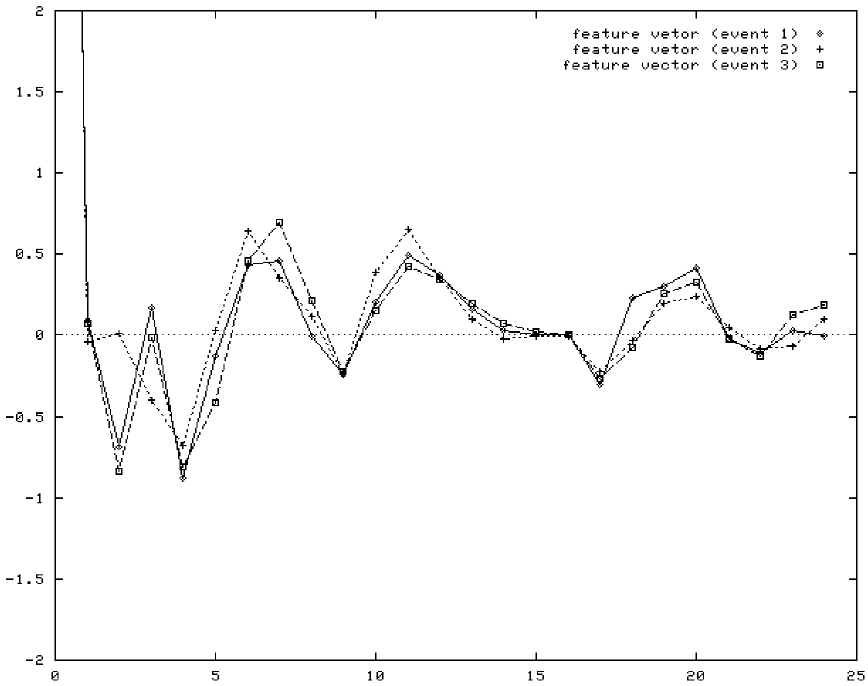

Figure 6.9: three feature vectors representing three instances of the same event.

In Fig. 6.10 three examples are reported of three different events. As we can see, features from different events show a different course, almost on every point of feature vector.

These feature vectors are eventually used in the learning phase: during the training stage, each event $\mathbf{x}$ in the training set is simply given by the supervisor one of the n possible labels $y_1, ..., y_n$ fixed beforehand. An OSH (Optimal Separating Hyperplane) is then computed between the classes, and the collection of obtained hyperplanes is utilized in the last phase, the *testing* of the system.

6.1.4 Experimental results

In its current version, *VIDERE* is able to detect the visual dynamic events in real time, but both the representation and the recognition stages are still computed offline. Therefore, in order to evaluate the performance of the system we acquired a number of image sequences, computed the representation for all the events detected in real time, and divided the obtained events to form the training and test sets.

VIDERE has been trained to recognize eight different actions taking place down a corridor leading to our Lab (see Figure 6.1). Examples of these actions are *somebody is walking up the corridor*

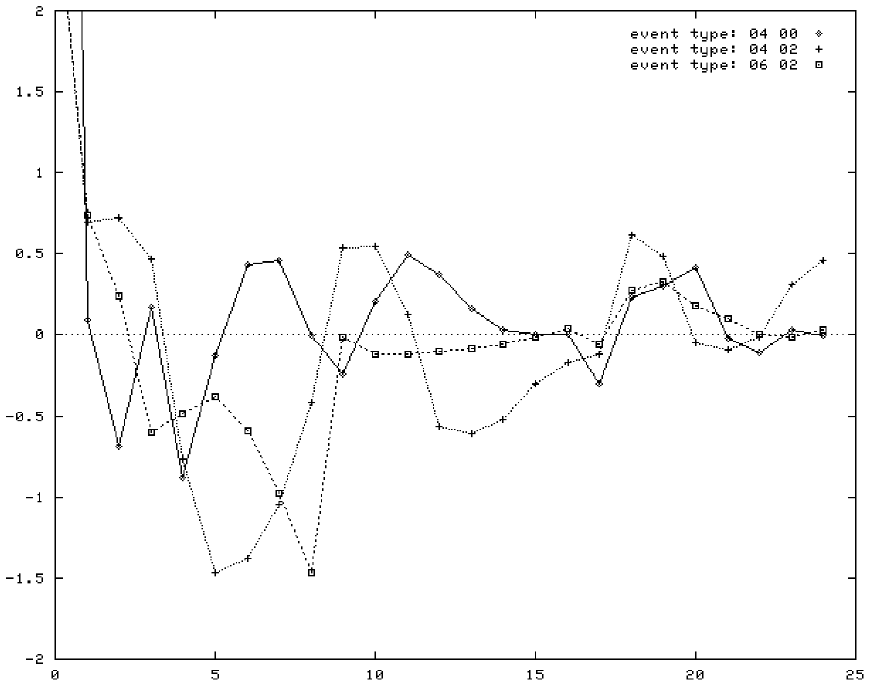

Figure 6.10: three feature vectors representing three different events.

and entering the last door on the left, and *somebody is exiting from room A, walking up the corridor and entering room B* The shortest event lasted for 15 seconds, the longest for 42. Four different peoples were involved as actors.

Given a total of 104 events (on average 13 examples for each class), each represented by a 25-dimensional vector, we checked the performance of *VIDERE* by means of the *leave-one-out* technique. This means that we trained the SVM for classification with 103 example and use the left out for testing the system for all the 104 possible combinations.

Several experiments have been done using different regression and classification methods. In the table 6.1 the results are reported. Let us briefly explain the abbreviation used to describe the experiments. For the regression phase:

- *Spl*: means that a one-dimensional spline kernel has been used with 10 knots and a low-pass filtering as described before.
- *Ave*: means that no regression has been performed. Instead, each vector **u**, **v**, and **a** has been divided into 9 partitions, and averaged on each partition. The final feature vector is constituted by the $9 \cdot 3$ average values and the elapsed time.
- *Last*: as before, but instead of the average for each partition we take the value of the last element.

For the classification phase we have:

- *Quad*: *quadratic* kernel in each of the 8 splines coefficients for the three measures and linear for the 25th component (elapsed time).
- *L(a),L(x),L(y)*: *linear* kernel respectively on the *area, x coordinate, y coordinate* component.
- *L(a,x,y)*: *linear* kernel on everything.

The results are reported in table 6.1.

Experiment	*Regression*	*Classification*	*Score*
1	*Spl*	*Quad*	98
2	*Spl*	$L(1)$	56%
3	*Spl*	$L(a)$	86%
4	*Spl*	$L(x)$	78%
5	*Spl*	$L(y)$	94%
6	*Ave*	$L(y)$	97%
7	*Ave*	$L(a, x, y)$	98%
8	*Last*	$L(a, x, y)$	95.5%

Table 6.1: Results of the *leave one out* testing method.

Since this is a multiclass recognition task, the obtained result is bound to depend on the order in which the various classifications take place (or are organized). In the case of the *tennis tournament* this means that the final result depends on the initial positions on the table of the various players, or classes. *A posteriori* analysis of the data showed that in all cases the classification was order independent which means that the system accuracy in distinguishing between two classes is very high (99.8%). Interestingly, on about 76% of the testing, the classified point lied outside the margin. This suggests that the chosen representation, though exceedingly simple, is sufficient for the task at hand. We can observe that:

- for this particular problem, it seems that the y-coordinate be very important for describing the considered dynamic events
- in the classification phase, the simple *linear kernel* works very well, achieving good success percentages in the tests
- the regression phase seems not to improve the overall performances. However we have to stress that in this application the regression is equivalent to a furter filtering module, thus it is useful just for "smoothing" the event description.

Observation As we have already pointed out, the *tennis tournament* method is order-dependent. Is interesting remarking that experimentally the classification turn out to be order-independent, while the *misclassification* often is order-dependent. In plain word, it means that if, say, a vector x appears to belong to class "a" after a tournement (and it's wrong), then after a different tournement might come out as belonging to class "b" (wrong again but different). This makes sense, because is very likely that the considered vector lies near to the margin of some of the hyperplanes.

When it happens that a misclassification is not order-dependent we have a strong indication that the training process has been biased. In fact most of the times, after a check on the initial data, it came out to be a supervisor's mistake rather than a classification error.

6.2 Gesture Recognition

6.2.1 Problem description

We applied the *VIDERE* system to another kind of visual task: recognizing hand and arm gestures. Since our goal is merely to test the performance of SVM's onto different applications, we approached a simple problem: discriminating among three types of gestures. In the figures 6.13 6.12 6.11 we can see an example of the three gestures: basically someone standing right in front of the camera moves its left arm to three different spatial position:

1. *right to left, down (1)* (fig. 6.11)
2. *right to left (2)* (fig 6.12)
3. *right to left, high (3)* (fig 6.13)

The low-level processing are very much like those described for the former application, but they are simpler: there is no background adjustment, nor a Kalman filtering stage. We decided to keep the image processing as poor as it was possible, in order to present the *SVM*-based classifier with very raw data, and thus verifying its flexibility. The low-level processing module provides the system with the binary image of a simple moving mask computed by taking the difference between the current frame and a background (assumed fixed).

6.2.2 Event Description

The binary image is analyzed, and the ellipsoid enclosing the moving, non-isolated pixels is extracted. The feature that we decided to select from the image sequence containing the event are then:

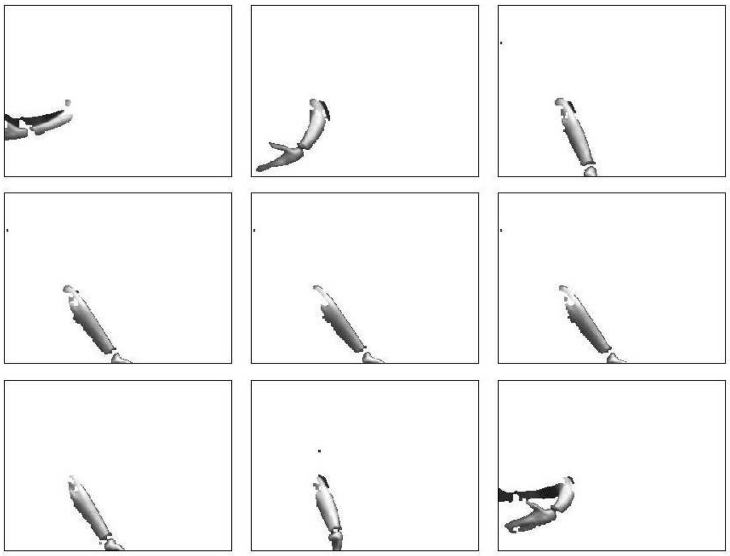

Figure 6.11: Event of type (1)

1. the *sine* and the *cosine* ($sin(\theta)$, $cos(\theta)$) of the angle (θ) between the major inertia-axis of the ellipsoid and the y-axis (see fig. 6.14),
2. the *ratio* between the major (a) and the minor (b) axis of the ellipsoid,
3. the *product* $\sqrt{a \cdot b}$

The feature vector describing the three events are constructed by partitioning the raw feature vector into 5 intervals, and then taking the five average values onto these intervals. Therefore the resulting feature vector has $5 \cdot 4 + 1$ dimensions (5 for each of the four considered parameters, plus the elapsed time). In Fig. 6.15 four instances of the same event are reported.

6.2.3 Experimental Results

The test was conducted using as a classifier a linear kernel, and performing a tennis tournement between the resulting classifiers. The number of available events was 105. The test performed states that the score achieved by *VIDERE* in recognizing these three gestures is about 96%.
The result is rather good if we remind that:

1. the raw data are quite noise-corrupted, since the image processing has been kept very primitive and there is no filtering,
2. the regression has been substituted by a simpler averaging routine,

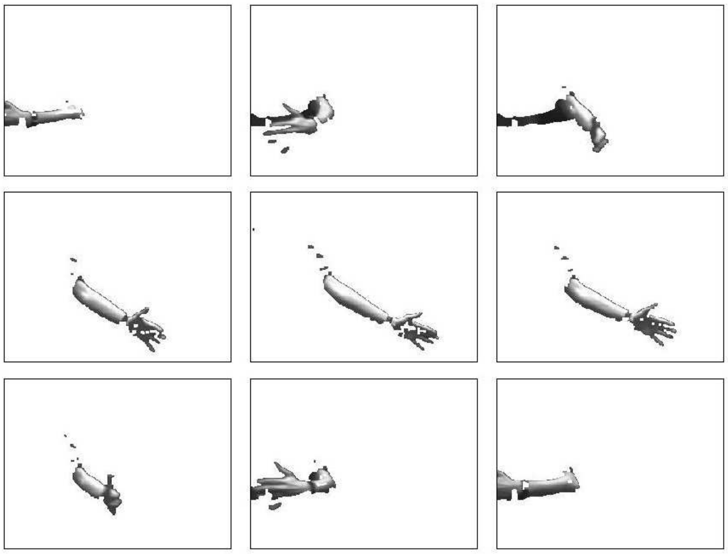

Figure 6.12: Event of type (2)

3. the classifier core is a linear kernel.

Let us remind again that experimental data has been collected asking some people (three different subjects) for doing one of the three possible movements. The same gesture from different people is different, thus produces a different representation. This can make very difficult the problem resolution when the number of available events is low. Despite this difficult the system turned out to be robust and reliable.

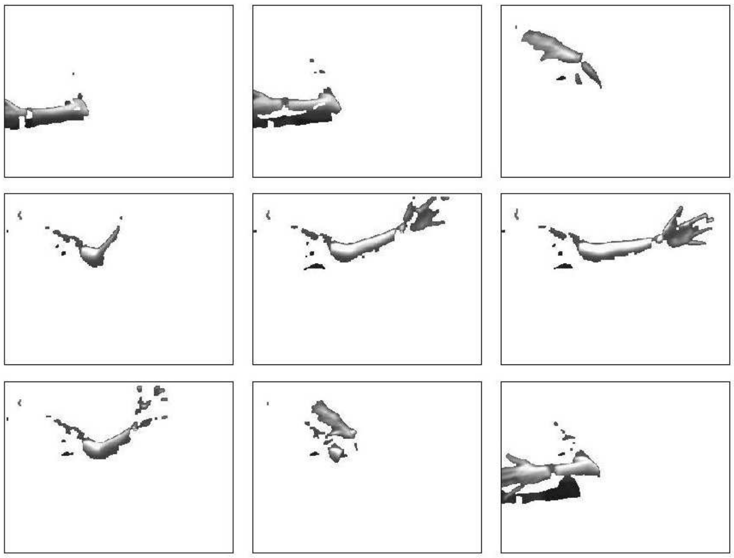

Figure 6.13: Event of type (3)

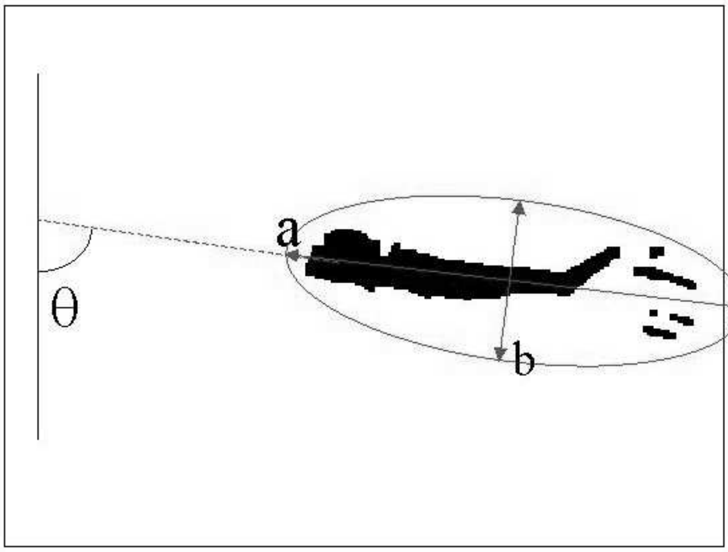

Figure 6.14: Description of the parameters used to describe the dynamic event.

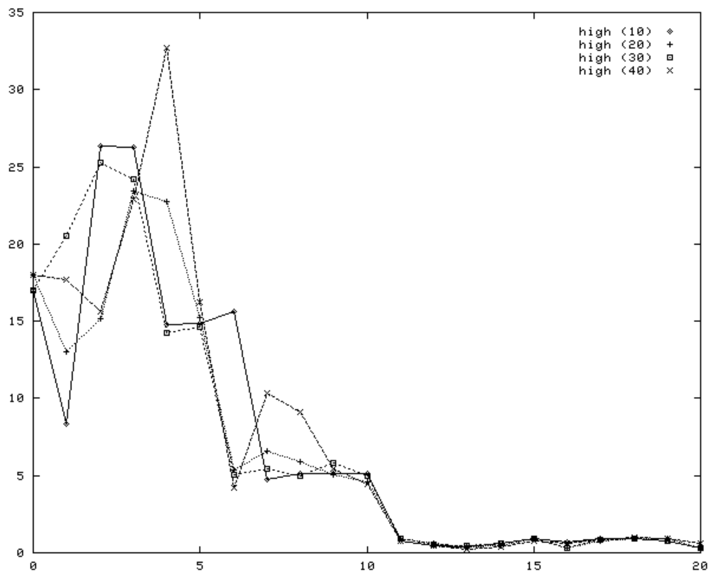

Figure 6.15: Four instances of the event “hight”.

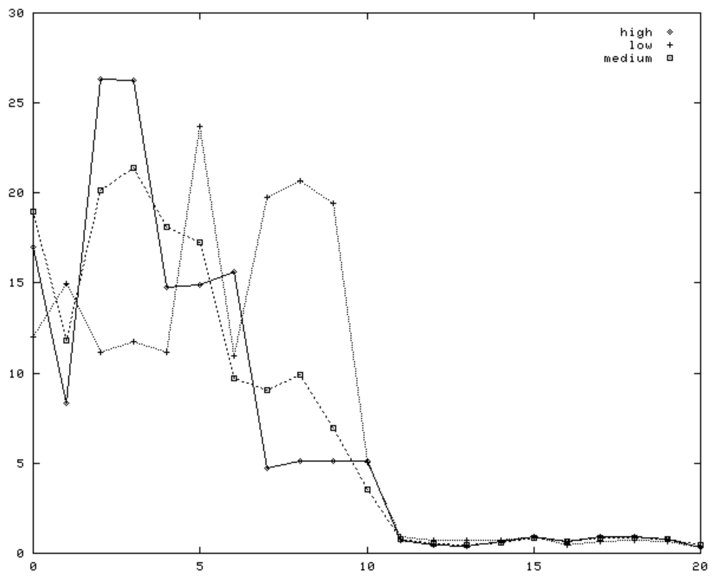

Figure 6.16: Three events examples, one for each possible class.

Chapter 7

Conclusions and Future Work

In this thesis we studied the *Support Vector Machines* (SVM) in the framework of Statistical Learning theory. We introduced the system *VIDERE* (VIsual Dynamic Event REcognition), which we have developed using the SVMs for solving event-related Computer Vision tasks. The system is composed by a *low-level* module, depending on the particular application, and a *high-level* module which provides the event description and classification, both using SVMs (independently from the application).
The system is *trainable*, thus it has to be provided with a set of examples (*trainig set*). For evaluating the system performances we considered two Computer Vision problems, namely:

- detecting and recognizing people motion
- recognizing simple gestures.

The system performed well in both the applications, with reliability and a high success rate in all the experiments, even with very small *training sets*. The results obtained have shown the power of SVM classification and indicated their robustness against noise.

Future Work
As current applications we are improving the system *VIDERE* for real-time multiple parallel event detection and recognition. As a further improvement, the system, after a brief *supervised* training, will be able to work *unsupervised*, collecting and clustering the dynamic events autonomously.

Bibliography

[B&S79] M.Bazaraa and C.M.Shetty
Nonlinear Programming
J.Wiley, New York, 1979.

[C&H53] R.Courant and D.Hilbert
Methods of Mathematical Physics
J.Wiley, New York, 1953.

[K&T61] A.N.Kolmogorov and V.M.Tihomirov
ϵ-Entropy and ϵ-Capacity of Sets in Function Spaces
in American Mathematical Society Translations, **17**, pp. 277-364, 1961.

[OFG97] E.Osuna, R.Freund and F.Girosi
An Improved Training Algorithm for Support Vector Machines
in IEEE Workshop on Neural Networks and Signal Processing, 1997.

[SBSW99] B.Schölkopf, P.Bartlett, A.Smola and R.Williamson
Shrinking the Tube: A New Support Vector Regression Algorithm
Futura pubblicazione in: M.S.Kearns, S.A.Solla e D.A.Cohn editors, Advances in Neural Information Processing Systems 11, MIT Press, Cambridge, MA.

[S96] A.Smola
Regression Estimation with Support Vector Learning Machines
Master's Thesis, Physik Department, Technische Universität München, 1996.

[SSM98] A.Smola, B.Schölkopf and K.Müller
The connection between Regularization Operators and Support Vector Kernels
GMD FIRST, Berlin, 1998.

[V82] V.N.Vapnik
Estimation of Dependences Based on Empirical Data
Springer Verlag, New York, 1982.

[V95] V.N.Vapnik
The Nature of Statistical Learning Theory
Springer Verlag, New York, 1995.

[V98] V.N.Vapnik
Statistical Learning Theory
J.Wiley & sons, New York, 1998.

[V&C71] V.N.Vapnik and A.Ja.Chervonenkis
On the uniform convergence of relative frequencies of events to their probabilities
Theory Probab. Apl., **16**, pp. 284-280, 1971.

[V&C74] V.N.Vapnik and A.Ja.Chervonenkis
Theory of pattern recognition, in Russian
Nauka, Mosca, 1974.

[V&C81] V.N.Vapnik and A.Ja.Chervonenkis
Necessary and sufficient conditions for the uniform convergence of the means to their expectations
Theory Probab. Apl., **26**, pp. 532-553, 1981.

[V&C91] V.N.Vapnik and A.Ja.Chervonenkis
The necessary and sufficient conditions for consistency of the method of empirical risk minimization
Pattern Recogn. and Image Analysis, **1**, pp. 284-305, 1991.

[VP98] A.Verri and M.Pontil
Properties of support vector machines
Neural Computation, (in press), 1998.

[VGS97] V.N.Vapnik, S.E.Golowich and A.Smola
Support Vector Method for Function Approximation, Regression Estimetion, and Signal Processing
Neural Information Processing Systems, Vol. 9. MIT Press, Cambridge, MA, 1997.

[COOP86] W.S.Cooper
Use of Optimal Estimation Theory, in Particular the Kalman Filter, in Data Analysis and Signal Processing
Review of Scientific Instrumentation, Vol. 57,pp.2862-2869 (1986).

[FZ92] Z. Zhang and O. Faugeras
3D Dynamic Scene Analysis: A Stereo Based Approach
(Springer Berlin 1992).