Introduction to support vector machines and applications to computational biology
DRAFT

Jean-Philippe Vert

July 17, 2001

Contents

Chapter 1

Introduction

Support vector machines (SVM) is a family of learning algorithms which is currently considered as one of the most efficient method in many real-world applications. The theory behind SVM was developed in the sixties and seventies by Vapnik and Chervonenkis, but the first practical implementation of SVM was only published in the early nineties. Since then the method gained more and more attention among the machine learning community thanks to its ability to outperform most other learning algorithms (including neural networks on decision trees) in many applications. As a result it has been successfully applied to all sorts of classifications issues, ranging from handwritten character recognition to speaker identification or face detection in images.

Recently SVM have been applied to biological issues, including gene expression data analysis or protein classification. Some claim that biological data mining applications are one of the most promising uses of SVM, particularly for the high dimensionality of the data. As a result research about SVM and computational biology is the object of much effort today, mainly due to researcher coming from the machine learning community. One can expect SVM to become a standard tool for bioinformaticians in the near future, just like clustering algorithms or dynamic programming methods today.

However, it is nowadays difficult for non-specialists to learn about SVM. Current tutorials or introductions to this topics remain full of not-so-easy mathematics which might dissuade many bioinformaticians from reading them. The goal is these seminar notes is to make SVM easy for non-specialists and thereafter to encourage their use in computational biology.

Chapter 2

A quick FAQ

In this chapter we give some basic definitions and try to answer simple questions about support vector machines (abbreviated SVM):

- What is a SVM?
- Why is it so popular?
- Why is it so efficient?
- What are the main results of Vapnik's statistical learning theory?
- What is the link between statistical learning theory and SVM?
- Why is it relevant to bioinformatics?

2.1 What is a SVM?

Support vector machines are a family of learning algorithms. The SVM we will study in these notes learn how to classify objects into two classes, based on a series of observations (this task is also called *pattern recognition*).

Let us call $\mathcal{X}$ the set of objects one might one to classify. A particular object $\vec{x}$ is an element of this set, which we denote by $\vec{x} \in \mathcal{X}$. The class or category of an object can take two values, which can be for instance -1 or $+1$. We will use the notation y to represent such a class, hence $y \in \{-1, +1\}$.

An observation is simply an object $\vec{x}$ together with its class y, which we denote by $(\vec{x}, y)$. With these notations a series of N observations can be written as follows:

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\},$$

where for any $i = 1, \ldots, N$, $(\vec{x}_i, y_i)$ is an observation.

The goal of a learning algorithm is to use a set of observations $\mathcal{S}$, which we call the *training set*, to learn a rule which can be used in the future to classify any new object $x \in \mathcal{X}$ into a class $y \in \{-1, +1\}$. SVM is one particular

algorithm which perform this pattern recognition task, i.e. learning from $\mathcal{S}$ a classifier for future objects.

Example 1 *Consider the problem of guessing whether a protein is an enzyme or not based on its amino-acid sequence (primary structure). In that case the set $\mathcal{X}$ is the set of all possible finite-length amino-acid sequences, and the class y would be $+1$ if the protein is an enzyme and -1 if it is not. A training set $\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ would be a database of N protein sequences $\vec{x}_1, \ldots, \vec{x}_N$ together with their classes $y_1, \ldots, y_N$. A SVM would then learn from this database a rule which could be used to classify any new protein as enzyme or non-enzyme from its primary structure.*

2.2 Why is it so popular?

The problem of learning a classification rule from a set of examples is very classical, and is the main focus of the discipline called *machine learning*. Many learning algorithms existed before the invention of SVM: famous methods include Fisher's discriminant, classification and regression trees (CART) or neural networks. However, since its introduction in the mid-nineties the SVM method has quickly become one of the most popular learning algorithms in the machine learning community, and is currently applied to more and more real-life classification problems (including biological issues).

The main reason for its popularity is that it is very efficient compared to other methods in many real-world application. Benchmark experiments have been conducted to compare different learning methods in various applications, and SVM are usually among the best algorithms, or even outperform other methods (for instance in text classification, written character recognition of protein family identification from the primary structure).

Being efficient means several things which we can quickly sum up:

- *Good generalization performance*: once the SVM is presented with a training set, it is able to learn a rule which can *correctly classify any new object* quite often.
- *Computational efficiency*: the algorithm is efficient in terms of speed and complexity, it is equivalent to looking for the minimum of a convex functional, i.e. with no local minima (unlike neural networks).
- *Robust in high dimensions* : dealing with large dimensional objects (like images of gene expression data) is usually difficult for learning algorithm, because of the overfitting issue (see next session). SVM seem to be more robust than other methods in such cases.

Finally SVM became also popular in the machine learning and mathematics community because it is based on a beautiful theory which is the subject of much theoretical research.

2.3 Why is it so efficient?

This is a difficult question, and today nobody knows for sure why it works so well. However theoretical results suggest that its efficiency is mainly due to its capacity to find rules which classify objects with high confidence, to prevent them form *overfitting*.

Overfitting is a central issue for learning algorithm. When a training set is presented to a learning algorithm, the algorithm usually tries to find a rule which explains well the observations, i.e., which correctly classifies most of the objects in the training set. Sometimes the algorithm can find a very complicated rule which perfectly classifies the objects in the training set, but this rule could be useless to classify new observations because it is too related to the training set: we say that such a rule does not *generalize* well, and this phenomenon is called overfitting.

Example 2 *Suppose the goal is to learn whether a protein is an enzyme or not based on the primary sequence, and the training set is very small, e.g. the following:*

$$\mathcal{S} = \Big\{ (MKSRAAVA\ldots,+1), (MNVMGFAA\ldots,+1), \\ (MKTRDSQS\ldots,-1), (MKNEKRKT\ldots,-1) \Big\}. \quad (2.1)$$

Hence this training set contains two enzymes and two non-enzyme proteins. From this training set one can imagine many rules which correctly classify all four sequences, e.g.:

> *If the sequence starts with MKSR or MNVM then the protein is an enzyme, otherwise it is not.*

Obviously this rule is not very good, because it won't be able to correctly classify many enzymes which are not in the training set, even though it correctly classifies every sequence in the training set: hence this rule will not generalize well, and overfitting occurs.

Example 3 *Neural networks are known to be very powerful to correctly classify almost any training set. However they often overfit, and many people consider them as "black box" which can learn almost anything but don't always generalize well. To prevent a neural network from overfitting, several methods are used to prevent it from "learning too well". For instance it is common to stop the iterative learning of a neural network before convergence occurs.*

Most learning algorithms try to find a rule which correctly classify the objects in the training set: this is the most natural way to learn. However it is usually not clear whether a learned rule will generalize well, or whether it will overfit.

To cope with this overfitting issue, Vapnik and Chervonenkis studied the link between the ability of a learning algorithm to learn a good rule for the training

set, and its ability to generalize well, i.e. not to overfit. Their work resulted in a theory called *statistical learning theory*, and their theoretical results then gave birth to SVM. As a result SVM were designed from the beginning with a single goal: *generalize well*, to the contrary of other learning algorithms which are designed with a different goal : *correctly classify the training set*.

2.4 What are the main results of Vapnik's statistical learning theory?

Statistical learning theory makes a link between two important features of a learning algorithm:

- its ability to learn a rule which correctly classifies most examples in the training set;
- the ability of the resulting rule to correctly classify new objects (i.e., the ability to generalize well)

Let us use the symbol f to denote a classification rule (or *classifier*), i.e. a mapping from the space of objects $\mathcal{X}$ to the space of classes $\{-1, +1\}$. In other words the classifiers f classifies any object $\vec{x} \in \mathcal{X}$ into the class $f(\vec{x})$, which is -1 or $+1$.

With these notations we can reformulate the learning task as follows : a learning algorithm uses a training set $\mathcal{S}$ to learn a particular classifier $f \in \mathcal{F}$. Here $\mathcal{F}$ denotes the set of all possible classifiers the algorithm can chose among. For example, in the case of decision trees, the set $\mathcal{F}$ can be seen as a set of trees with questions attached to each node.

In order to chose a classifier f from the set $\mathcal{F}$ based on the training set $\mathcal{S}$, it is natural to judge the quality of every $f \in \mathcal{F}$ by their ability to correctly classify the objects in the training set. Hence we define the *empirical risk* of the classifier f as the percentage of good classification it makes on the training set. Let us denote by $R_{emp}(f))$ this empirical risk, which is therefore a number between 0 and 1: for a classifier f which correctly classifies all examples in the training set, we have $R_{emp}(f) = 0$, while for a "bad" classifier f which makes a mistake for each example, we have $R_{emp}(f) = 1$.

Hence $R_{emp}(f)$ characterizes the capacity of a rule f to correctly classify the examples in the training set. Classical learning algorithms like neural network usually search for classification rules $f \in \mathcal{F}$ with $R_{emp}(f)$ as small as possible.

What about the generalization performance of a rule f? A convenient way to define it is to suppose that the training examples (as well as the future examples to be classified) are generated one by one (and independently) by a random machine according to a fixed probability distribution. Under this hypothesis an object to be classified is a random object X[1], its class is a random variables

[1]The space of objects is supposed to satisfy all conditions required to define a random object

which can take only two values (for instance +1 and -1), and an observation (X, Y) is governed by a probability P. In this probabilistic framework, the probability P is of course unknown *a priori*. The only thing we assume is that the training set is made of N random variables $(X_1, Y_1), \ldots, (X_N, Y_N)$ which are generated according to P.

It is now possible to define precisely what "generalization performance" of a rule f means: it is the probability that the rule f makes a mistake on a new sample randomly generated according to the distribution P, which we can write as:

$$R(f) = P\left(f(X) \neq Y\right).$$

$R(f)$ is called the *risk* of the classifier f, and quantifies the ability of a rule to generalize well: if $R(f) = 0$ then f will never make any error of any new observation, so it generalizes perfectly.

For a given rule f and a given training set $\mathcal{S}$, the empirical risk $R_{emp}(f)$ can be observed but the risk $R(f)$ is not observed. Intuitively however, it is natural to think that in many cases, if $R_{emp}(f)$ is small (i.e., if the rule f makes few errors on the training set), then $R(f)$ is small too. This is why many learning algorithms try to find a rule f with a small $R_{emp}(f)$, in the hope that $R(f)$ will be small too.

This intuition is partially true: for a given rule f, by the law of large numbers, the empirical risk $R_{emp}(f)$ converges to the risk $R(f)$ when the size of the training set increases, so $R_{emp}(f)$ is a good indicator of $R(f)$.

The situation is a bit more complex when one analyses a learning algorithm. Indeed a learning a algorithm has to *chose a rule* f from a set $\mathcal{F}$ based on the observation of the training set. As we said before, a natural choice for a learning algorithm is to chose the rule with smallest empirical risk on the training data, i.e. to chose the rule $\hat{f}$ defined by:

$$R_{emp}(\hat{f}) = \inf_{f \in \mathcal{F}} R_{emp}(f).$$

This method is usually called *empirical risk minimization (ERM)*. On the other hand the real goal of the learning algorithm is to find the rule which generalizes best, i.e. to find the rule f^* which satisfies:

$$R(f^*) = \inf_{f \in \mathcal{F}} R(f).$$

The central question now becomes: is it true that the risk $R(\hat{f})$ of the rule selected by empirical risk minimization is not far from $R(f^*)$? The answer to this question appears to be negative when overfitting occurs: in that case, one can find rules very good at explaining the data (i.e., $R_{emp}(\hat{f})$ is very small), but with a poor generalization capacity (i.e., $R(\hat{f})$ is not small, and better rules could be found).

The main results in statistical learning theory relate $R(\hat{f})$ to $R(f^*)$. A typical result is the following:

$$\mathbf{E}R(\hat{f}) \leq R(f^*) + c\sqrt{\frac{V(\mathcal{F})}{N}}, \tag{2.2}$$

where c is a universal constant, $V(\mathcal{F})$ is a quantity characteristic of the set of rules $\mathcal{F}$ called the *Vapnik-Chervonenkis dimension (or simply VC dimension)* of the class, and N is the number of examples in the training set. One should remember that the selected rule $\hat{f}$ depends on the training set, so it is in fact a random rule, and its risk $R(f^*)$ is therefore a random variable: this is why we use the symbol $\mathbf{E}$ in equation (2.2) to denote its average with respect to the random choice of the training sample.

Equation (2.2) shows that on average, the risk of the selected rule $\hat{f}$ is not far from the best possible risk $R(f^*)$ if two conditions hold:

- the number of observations N in the training set should be large enough;
- the VC dimension of the set of possible rules $\mathcal{F}$ should be small enough.

Hence the VC dimension plays a central role: the smaller the VC dimension, the better the generalization performance of the rule selected by empirical risk minimization. In other words, a learning algorithm will find a rule which generalizes well if it can chose the rules from a set $\mathcal{F}$ with small VC dimension.

It would be too long to study in detail what the VC dimension of a set is: let us just say that the "larger" a class $\mathcal{F}$, the larger its VC dimension. For example the set of rules a neural network can chose is very large, because it can almost always find a rule which perfectly explains any training set: not surprisingly its VC dimension tends to infinity when the number of neurons increases, which means that equation (2.2) makes no sense for a neural network.

In conclusion, the main contribution of statistical learning theory to the design of learning algorithm is to point out the importance of controlling the "size" of the set of rules $\mathcal{F}$ the algorithm can chose among, as measured by its VC dimension.

2.5 What is the link between statistical learning theory and SVM?

We saw in the previous section that the main results of the so-called statistical learning theory relate the risk of rule $\hat{f}$ selected by a learning algorithm from a set of rules $\mathcal{F}$ by empirical risk minimization to the risk of the best rule in the the set $\mathcal{F}$ by equation (2.2). Using this result, the fundamental idea behind SVM is the following: *in order to obtain good generalization performance, the VC dimension must be controlled.*

Controlling the VC dimension ensures good generalization performance. That means that the set $\mathcal{F}$ has to be "small". However, if it is too "small", then the rules is contains might be too simple to correctly classify the objects in the training set, simply because the classification task is not that easy! In other words, if $\mathcal{F}$ is small then there is a risk that $R(f^*)$ might be large. In that case, even if $R(\hat{f})$ is a good approximation of $R(f^*)$ (because the VC dimension is small), the resulting rule will not be good because $R(f^*)$ itself is so small.

Hence there are two opposite goals when designing a learning algorithm:

- Chose $\mathcal{F}$ as large as possible to ensure that at least one element in $\mathcal{F}$ has a small risk;
- Chose $\mathcal{F}$ as small as possible to ensure that the risk of the selected rule is almost as small as the risk of the best rule.

In order to solve this apparent paradox Vapnik proposed to consider a family of rules $\mathcal{F}$ which is the increasing union of families with increasing VC dimension, i.e.:

$$\mathcal{F}_0 \subset \mathcal{F}_1 \subset \ldots \subset \mathcal{F}_n \subset \ldots \subset \mathcal{F},$$

with:

$$V(\mathcal{F}_1) \leq V(\mathcal{F}_2) \leq \ldots \leq V(\mathcal{F}_2) \leq \ldots \leq V(\mathcal{F}).$$

In each family k let $\hat{f}_k \in \mathcal{F}_k$ denote the rule with the smallest empirical risk in the family of rules $\mathcal{F}_k$, and let $f_k^* \in \mathcal{F}_k$ denote the rule in $\mathcal{F}_k$ with smallest risk. When k increases, $R_{emp}(\hat{f}_k)$ decreases because $\hat{f}_k$ minimizes the empirical risk of a set of increasing size ($\mathcal{F}_k \subset \mathcal{F}_{k+1}$). On the other hand, the term:

$$c\sqrt{\frac{V(\mathcal{F}_k)}{l}}$$

in equation (2.2), which we call the confidence interval, increases with k, because $V(\mathcal{F}_k) \leq V(\mathcal{F}_{k+1})$.

As a result, the bound on the risk $R(f_k^*)$ given by equation (2.2) typically first decreases with k, and then increases as shown in Figure 2.5. As a result there might exist an optimal k^* which ensures the lowest upper bound for the generalization error of $\hat{f}_{k^*}$: choosing this $\hat{f}_{k^*}$ is called *structural risk minimization.*

A SVM is an implementation of the structural risk minimization principle. We will see more concretely in the next chapter what is the family of rules $(\mathcal{F}_k)_{k\geq 1}$ used by SVM, and how the minimization of the VC dimension is performed.

2.6 Why is it relevant to bioinformatics?

In computational biology or bioinformatics, many problems can be considered as classification problem. Let us just mention some of them:

- *gene finding in DNA*: the object $\vec{x}$ is a part of a DNA strand and the class is $+1$ if the nucleotide at the center of $\vec{x}$ is inside of a gene, -1 otherwise
- *sequence-based gene classification* : the object $\vec{x}$ is a gene sequence, and the class is a functional class from a functional hierarchy like for instance the MIPS hierarchy.
- *gene classification from microarrays experiment* : the object is the expression profiles of the genes, and the class is a functional class.

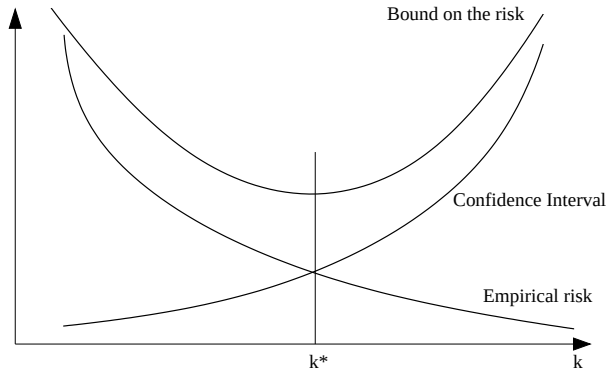

Figure 2.1: Structural risk minimization

- *protein secondary structure prediction* : the object is a sequence of amino-acid, the class is the local secondary structure
- *protein 3D structure prediction from sequence* : the object is a sequence of amino-acids, the class is a particular 3D fold.
- *protein localization in the cell* : the object is an amino-acid sequence, the class is a position in the cell (nucleus, membrane etc...)

Many more examples can probably be added to this list. Due to the importance of these issues in bioinformatics there is an urge to develop efficient methods to solve them : SVM appear to be the state-of-the-art methods in many other applications, so it is reasonable to think that this relatively new technology will generate interesting progress in the coming years in bioinformatics too.

Some researchers have already started to apply SVM to bioinformatics issues, and their results are usually very good compared to classical methods. As a result it is probably worth trying to master this technology today, as it will probably be a central theme of research for bioinformaticians in the near future if not now.

Chapter 3

Simplest SVM

We saw in the preceding chapter that a support vector machine is a learning algorithm whose purpose is to classify objects into two classes. Let us now study in more detail the algorithm itself.

The objects to be classified can be almost anything, like pictures, sound, molecules, graphs etc... In this chapter, however, we will only consider a particular kind of objects, namely finite dimensional real vectors. A m-dimensional real vector $\vec{x}$ has m coordinates which we can write $\vec{x} = (x_1, \ldots, x_m)$, where each x_i is a real number (i.e., $x_i \in \mathbb{R}$ for $i = 1, \ldots, m$). In that case the set of possible objects is denoted by $\mathcal{X} = \mathbb{R}^m$.

Observe that such objects are often derived from more complex objects (like images, molecules...) by computing the values of different features to characterize the object.

Example 4 *Suppose one wants to represent proteins for classification. One possibility is to create a 20-dimensional real-valued vector for each protein, where each coordinate represent the percentage of a particular amino acid in the protein composition.*

The advantage of studying such objects is that they represent points in a Euclidean space. It is possible to design simple linear classifiers on such spaces, and we will meet in this chapter our first SVM, namely a particular linear classifier.

3.1 Linear classifiers

Suppose $m = 2$, i.e. each object has only two coordinates: $\vec{x} = (x_1, x_2)$. Such an object can be represented by a point in a plan, see Figure 3.1

In that space a line can be characterized by a linear equation of the form:

$$\vec{w}.\vec{x} + b = 0, \tag{3.1}$$

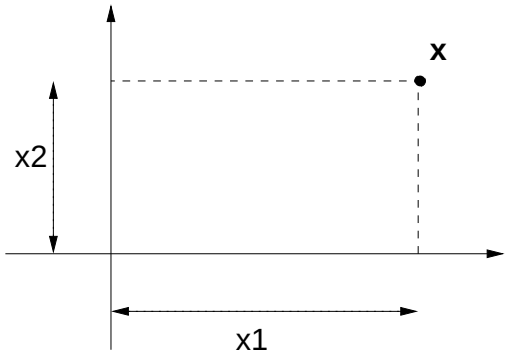

Figure 3.1: Vector representation in $\mathbb{R}^2$

where $\vec{w}$ and b are two vectors and $\vec{w}.\vec{x}$ is the dot product between $\vec{w}$ and $\vec{x}$, i.e.:

$$\vec{w}.\vec{x} = w_1 \times x_1 + w_2 \times x_2.$$

In other words the set of points $\vec{x}$ which satisfy equation (3.1) form a straight line in the plan, see Figure 3.1.

It is well known that two vectors $\vec{x}_1$ and $\vec{x}_2$ are orthogonal (or perpendicular) if and only if their dot product is equal to zero, i.e. $\vec{x}_1.\vec{x}_2 = 0$. This property can be used to check that the vector $\vec{w}$ is orthogonal to the line defined by equation (3.1). Indeed, if $\vec{x}_1$ and $\vec{x}_2$ are two points on the line, they satisfy $\vec{w}.\vec{x}_1 + b = \vec{w}.\vec{x}_2 + b = 0$. By subtracting the two inequalities one get $\vec{w}.(\vec{x}_1 - \vec{x}_2) = 0$, which shows that the vector $\vec{w}$ is orthogonal to the vector $\vec{x}_1 - \vec{x}_2$, which is precisely parallel to the line (see Figure 3.1).

Such a line divides the whole space into two areas, or *half-spaces*. One is defined by the set of $\vec{x}$ such that:

$$\vec{w}.\vec{x} + b > 0,$$

and the other is defined by the set of $\vec{x}$ such that:

$$\vec{w}.\vec{x} + b < 0.$$

The line itself is the frontier between the two areas (see Figure 3.1)

Such a line defines a classifier which we call a *linear classifier* as follows. It classifies any point $\vec{x} \in \mathbb{R}^2$ depending on its position with respect to the line : $\vec{x}$ is classified as $+1$ if $\vec{w}.\vec{x} + b > 0$, or as -1 if $\vec{w}.\vec{x} + b < 0$. If $\vec{x}$ falls on the line ($\vec{w}.\vec{x} + b = 0$) then there is indeterminacy.

This construction can be generalized to any dimension greater than 2 by keeping the same notations. In $\mathbb{R}^m$ the dot product is defined by:

$$\vec{w}.\vec{x} = w_1 \times x_1 + w_2 \times x_2 + \ldots + w_m \times x_m.$$

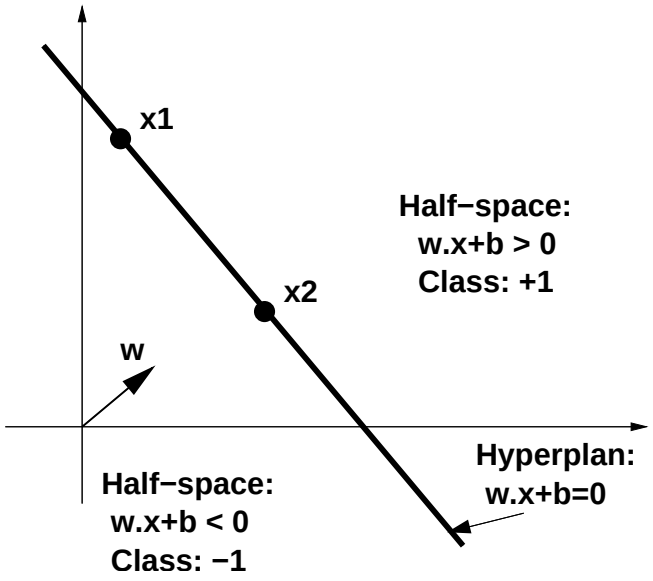

Figure 3.2: Linear separation in $\mathbb{R}^2$

For any vectors $\vec{w}$ and b in $\mathbb{R}^m$, the set of points $\vec{x}$ which satisfies :

$$\vec{w}.\vec{x} + b = 0$$

is called an hyperplane (the equivalent to a line when $m = 2$) which divides the whole space into two half-spaces. It therefore defines a linear classifier which classifies any vector $\vec{x}$ into one class or the other depending on the sign of $\vec{w}.\vec{x}+b$.

3.2 Linearly separable training set

Using the notations introduced in Chapter 2 let us consider a training set $\mathcal{S}$ which a set of points $\vec{x}_i \in \mathbb{R}^m$ together with their classes $y_i \in \{-1, +1\}$ for $i = 1, \ldots, N$, i.e.:

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}\,.$$

Our goal in this chapter is to learn a linear classifier from such a training set. Obviously linear classifiers are very simple, and they might not be very efficient to classify a given training set, e.g. if positive and negative examples are spread everywhere in the plan. Therefore we will only consider a very limited class of training set in this chapter, namely the training set which can be perfectly classified by at least one linear classifier. This restriction will enable us to present the main features of SVM, and will be discarded in the next chapter.

We say that the training set $\mathcal{S}$ is *linearly separable* if there exists at least one linear classifier defined by two vectors $\vec{w}$ and b which correctly classifies all

objects in $\mathcal{S}$, i.e.:

$$\begin{cases} \vec{w}.\vec{x}_i + b > 0 & \text{if } y_i = +1 \\ \vec{w}.\vec{x}_i + b < 0 & \text{if } y_i = -1 \end{cases}$$

for all $i = 1, \ldots, N$. This situation is represented in Figure 3.2. In this chapter, *we will only consider linearly separable training sets.*

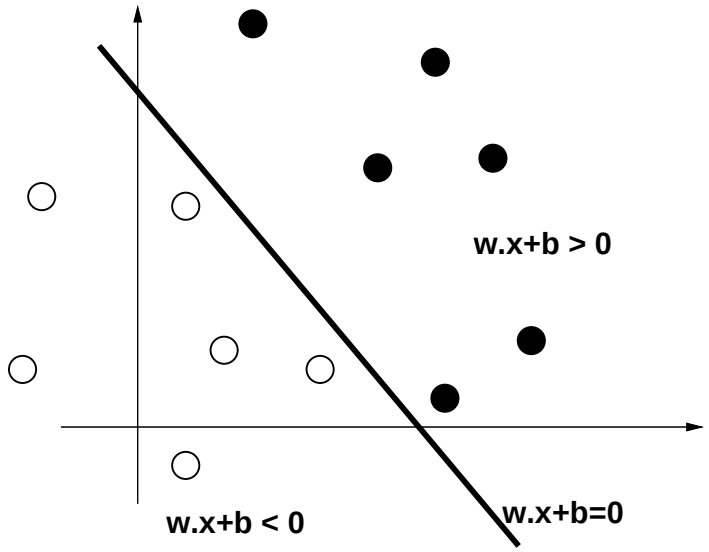

Figure 3.3: Separable training set : black circles are positive examples, white circles are negative examples

3.3 Linear SVM for separable training set

If the training set is linearly separable (see Figure 3.2) then there are usually many linear classifiers which correctly classify it (see Figure 3.3).

In order to apply the results of the statistical learning theory (see Chapter 2) remember that in order to chose a "good" classifier one should take care of two factors:

- chose a classifier with an empirical risk as small as possible;
- chose a classifier from a family with VC dimension as small as possible.

In the case of linear classifiers for a separable training set the first condition is fulfilled by choosing any classifier which correctly classifies all objects in the training set, i.e. by choosing any of the classifiers depicted in Figure 3.3.

In order to fulfill the second condition one needs to know what VC dimension means for families of linear classifiers. A classical result in learning theory states

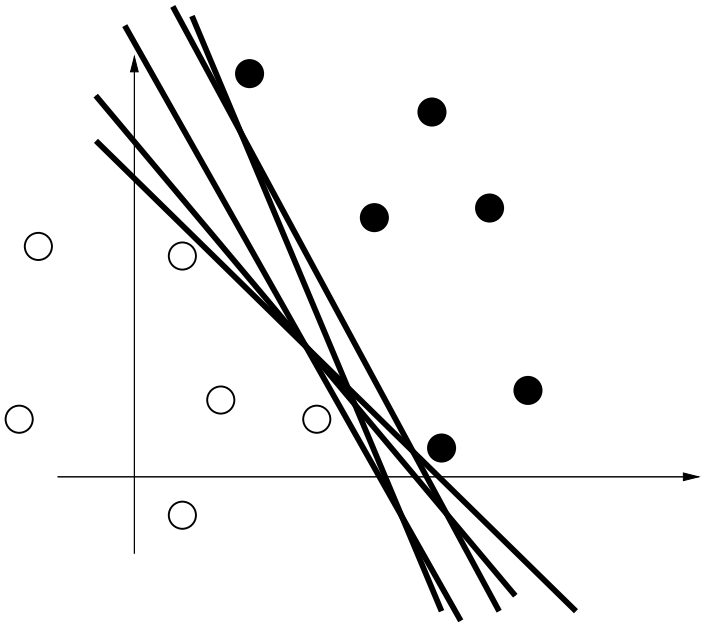

Figure 3.4: Different linear classifiers which correctly classify the training set

that in that case, the VC dimension is related to the smallest distance between a point in the training set and the separating hyperplane. This distance is also called *margin* and denoted by γ. Figure 3.3 shows the margin for a particular classifier.

More precisely the VC dimension of the set of linear classifiers which correctly classify the data with a margin at least equal to γ decreases with γ. As a result an application of the structural risk minimization principle leads to the following rule:

- Chose a linear classifier which correctly classifies all training examples (to have the smallest possible empirical risk);
- Among those classifiers, chose the one with the largest margin (to have the smallest possible VC dimension).

The hyperplane with the largest margin for a given training set is called the *optimal hyperplane*. The learning algorithm which chooses the liner classifier with largest margin is called a *linear support vector machine for separable training set*. This is the method we will focus on in the remaining of this chapter.

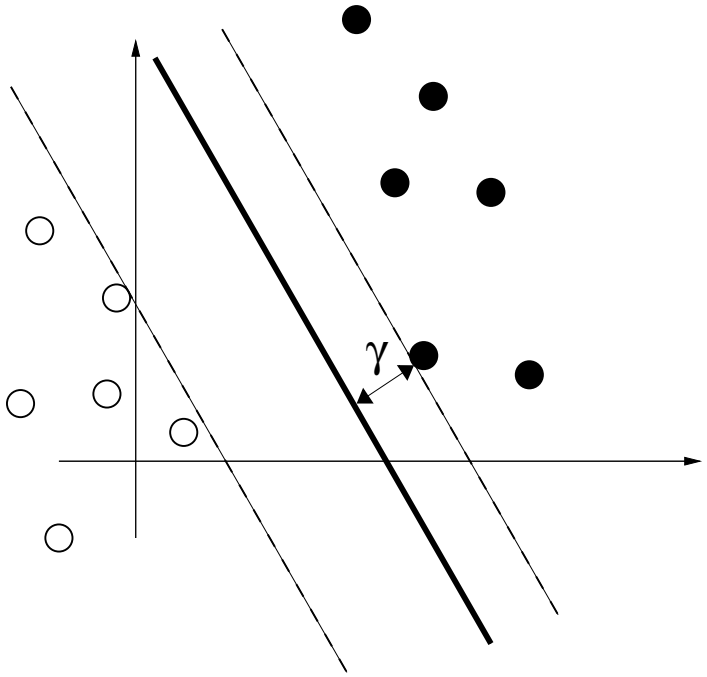

Figure 3.5: The margin γ of a linear classifier

3.4 Finding the optimal hyperplane

In this section we show how to compute explicitly the optimal hyperplane from a training set. In other words we describe the algorithm used by a linear SVM for separable training set.

Consider the set of pairs $(\vec{w}, b)$ which satisfy the following inequalities for any $i = 1, \ldots, N$ (i.e., for any example in the training set):

$$\begin{cases} \vec{w}.\vec{x}_i + b \geq 1 \text{ if } y_i = +1, \\ \vec{w}.\vec{x}_i + b \leq -1 \text{ if } y_i = -1. \end{cases} \tag{3.2}$$

Each such pair $(\vec{w}, b)$ defines a classifier which correctly classify the training set, because the corresponding classifier f is defined by :

$$f(x) = \begin{cases} +1 & \text{if } \vec{w}.\vec{x} + b > 0 \\ -1 & \text{if } \vec{w}.\vec{x} + b < 0 \end{cases}$$

Conversely, if a classifier separates the training data with a positive margin (e.g., like in Figure 3.3), then it can be represented by a pair $(\vec{w}, b)$ which satisfy inequality (3.2). As a result *Equation (3.2) characterizes the set of linear*

classifiers defined by the vector pairs $(\vec{w}, b)$ *which separate the training set with positive margin.*

In order to compute the particular pair $(\vec{w}^*, b^*)$ which achieves the largest margin we need to compute the margin achieved by any pair $(b, \vec{w})$ which satisfies inequality (3.2), and then minimize this functional.

Equation (3.2) says that there is no training point between the hyperplanes defined by the equations $\vec{w}.\vec{x} + b = -1$ and $\vec{w}.\vec{x} + b = 1$ (see Figure 3.4). As a

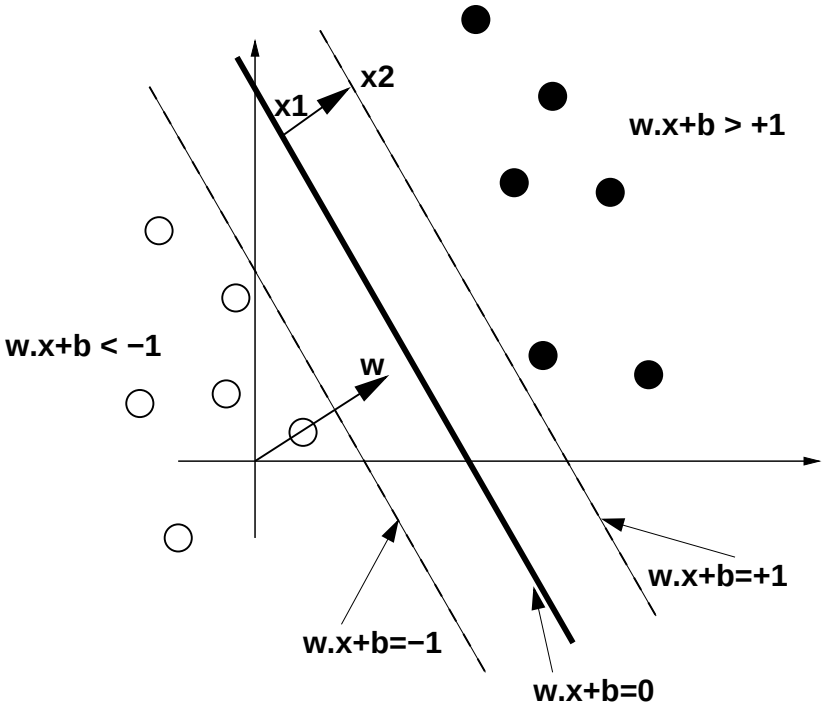

Figure 3.6: A separating hyperplane

result, the margin γ is at least equal to the distance between the hyperplanes defined by the equations $\vec{w}.\vec{x}+b = 0$ and $\vec{w}.\vec{x}+b = +1$, i.e. the distance between $\vec{x}_1$ and $\vec{x}_2$ on Figure 3.4. To compute this distance it suffices to write that:

$$\begin{cases} \vec{w}.\vec{x}_1 + b = 0, \\ \vec{w}.\vec{x}_2 + b = +1. \end{cases}$$

By subtracting the first equality from the second we get:

$$\vec{w}.(\vec{x}_2 - \vec{x}_1) = 1. \tag{3.3}$$

Now $\vec{x}_2 - \vec{x}_1$ is chosen to be orthogonal to the separating hyperplane, and $\vec{w}$ is also orthogonal to this hyperplane: as a result $\vec{w}$ and $\vec{x}_2 - \vec{x}_1$ are parallel, and

therefore the dot product satisfies:

$$|\vec{w}.(\vec{x}_2 - \vec{x}1)| = ||\vec{w}|| \times ||\vec{x}_2 - \vec{x}_1||,$$

where $||\vec{w}||$ denotes the norm of the vector $\vec{w}$. Using this equation with equality (3.3) finally leads to:

$$||\vec{x}_2 - \vec{x}_1|| = \frac{1}{||\vec{w}||}.$$

In other words the distance between the hyperplanes defined by the equations $\vec{w}.\vec{x} + b = 0$ on the one hand and $\vec{w}.\vec{x} + b = 1$ on the other hand is exactly equal to $1/||\vec{w}||$. Since the margin of the corresponding classifier is always larger than this distance (see Figure 3.4), we see that the smaller $||\vec{w}||$, the larger the margin. As a result the problem of searching for the optimal hyperplane can be restated as follows:

> The optimal hyperplane is defined by a pair $(\vec{w}, b)$ which satisfies:
>
> $$\begin{cases} \vec{w}.\vec{x}_i + b \geq 1 \text{ if } y_i = +1, \\ \vec{w}.\vec{x}_i + b \leq -1 \text{ if } y_i = -1. \end{cases}$$
>
> and for which the norm $||\vec{w}||$ is minimum.

This is a constrained optimization problem, whose solution is the optimal hyperplane.

3.5 Solving the optimization problem

Let us reformulate the optimization problem in a more classical form:

> Minimize
>
> $$||w||^2$$
>
> under the constraints:
>
> $$\text{For } i = 1, \ldots, N, \qquad y_i(\vec{w}.\vec{x}_i + b) - 1 \geq 0.$$

This is the same problem as the one presented in the preceding section, because minimizing $||\vec{w}||$ is the same as minimizing $||\vec{w}||^2$, and because the constraints are the same.

This appears to be a very classical problem in mathematics : minimizing a quadratic functional under linear constraints. Such a problem is called *quadratic program*, and most optimization softwares include procedures to automatically solve them. However it is worth studying a bit more this problem for at least three reasons:

- We will be able to give a different formulation of this problem (the so-called *dual formulation*) which will express the importance of each example in the training set. The most important of them will be called *support vectors*.

- The implementation of the dual formulation is usually more efficient than the direct implementation of the minimization problem.

- The dual formulation will generalize to non-separable training sets (Chapter 4) and to non-linear SVM (Chapter 5).

Using a classical approach (explained in as an appendix in Chapter 7) to solve constrained minimization problems, let us introduce the *Lagrangian* function, defined for any vector $\vec{w}$, real number b and dual vector $\vec{\lambda} = (\lambda_1, \ldots, \lambda_N)$ by:

$$L(\vec{w}, b, \vec{\lambda}) = ||\vec{w}||^2 - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\vec{w}.\vec{x}_i + b \right) - 1 \right].$$

In Chapter 7 we show that to formulate the dual problem we first need to compute, for each positive λ, the values $(\vec{w}_{\vec{\lambda}}, b_{\vec{\lambda}})$ which minimize the Lagrangian (with $\vec{\lambda}$ being fixed). To do this we can just differentiate the Lagrangian with respect to $\vec{w}$ and b and find the values for which they are equal to zero. Therefore we compute:

$$\frac{\partial L}{\partial \vec{w}}(\vec{w}, b, \vec{\lambda}) = 2\vec{w} - \sum_{i=1}^{N} \lambda_i y_i \vec{x}_i = \vec{0},$$

and

$$\frac{\partial L}{\partial b}(\vec{w}, b, \vec{\lambda}) = -\sum_{i=1}^{N} \lambda_i y_i = 0.$$

From the first equation we see that for any $\vec{\lambda}$, the vector $\vec{w}_{\vec{\lambda}}$ can be expressed as:

$$\vec{w}_{\vec{\lambda}} = \frac{1}{2} \sum_{i=1}^{N} \lambda_i y_i \vec{x}_i. \tag{3.4}$$

From the second equation we can not directly express $b_{\vec{\lambda}}$ in terms of $\vec{\lambda}$, but we have a constraint on $\vec{\lambda}$:

$$\sum_{i=1}^{N} \lambda_i y_i = 0. \tag{3.5}$$

For any $\vec{\lambda}$ such that this constraint is not fulfilled, the minimum of the Lagrangian $(\vec{w}, b) \to L(\vec{w}, b, \vec{\lambda})$ is equal to $-\infty$ (by taking $b = +\infty$ if $\sum_{i=1}^{N} \lambda_i y_i > 0$ and $b = -\infty$ if $\sum_{i=1}^{N} \lambda_i y_i < 0$).

On the other hand, for any $\vec{\lambda}$ which satisfies the condition in equation (3.5) we can use equation (3.4) to compute:

$$\inf_{\vec{w},b} L(\vec{w},b,\vec{\lambda}) = L(\vec{w}_{\vec{\lambda}}, b_{\vec{\lambda}}, \vec{\lambda}) \tag{3.6}$$

$$= ||\frac{1}{2}\sum_{i=1}^{N}\lambda_i y_i \vec{x}_i||^2 - \sum_{i=1}^{N}\lambda_i \left[y_i \left(x_i \sum_{j=1}^{N}\lambda_j y_j \vec{x}_j + b_\lambda \right) - 1 \right] \tag{3.7}$$

$$= \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j - b\sum_{i=1}^{N}\lambda_i y_i + \sum_{i=1}^{N}\lambda_i \tag{3.8}$$

$$= -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j + \sum_{i=1}^{N}\lambda_i \tag{3.9}$$

For any vector $\vec{\lambda}$ let us call $W(\vec{\lambda})$ the minimum value of the Lagrangian when $\vec{\lambda}$ is fixed and $(\vec{w}, b)$ can vary without any constraint, i.e.:

$$W(\vec{\lambda}) = \inf_{\vec{w},b} L(\vec{w},b,\vec{\lambda}).$$

From the computations above we see that $W(\vec{\lambda})$ can be expressed as follows for any vector $\vec{\lambda}$:

$$W(\vec{\lambda}) = \begin{cases} -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j + \sum_{i=1}^{N}\lambda_i & \text{if } \sum_{i=1}^{N} y_i \lambda_i = 0, \\ -\infty & \text{otherwise.} \end{cases} \tag{3.10}$$

As explained in Chapter 7 the dual problem is the problem of finding a *positive* dual vector $\vec{\lambda}^*$ which *maximizes* the function $W(\vec{\lambda})$. From the expression of $W(\vec{\lambda})$ in equation (3.10) we see that λ^* has to satisfy the constraint:

$$\sum_{i=1}^{N} y_i \lambda_i = 0,$$

in which case we can use the formulation of equation (3.10) which shows that $W(\vec{\lambda})$ is a quadratic function.

If we call $\vec{\lambda}^*$ the solution of the dual problem (i.e., $\vec{\lambda}^*$ is the positive vector which maximizes $W(\vec{\lambda})$), then we can recover $\vec{w}^*$ using equation (3.4). This gives the direction of the optimal hyperplan. However b^* can not be obtained directly from $\vec{\lambda}^*$. It can be obtained by $\vec{w}^*$, because for a given hyperplan direction it is easy to find the one with the largest margin. It is defined by:

$$\begin{cases} \min_{i:y_i=+1} \vec{w}^*.\vec{x}_i + b^* = +1, \\ \max_{i:y_i=-1} \vec{w}^*.\vec{x}_i + b^* = -1. \end{cases}$$

(the first equation means that all positive training example satisfy $\vec{x}.\vec{w}^*+b^* \geq 1$, and that at least one of them is on the hyperplan $\vec{x}.\vec{w}^* + b^* = 1$; the second equation is equivalent for negative training examples). Summing these two equations we obtain:

$$\min_{i:y_i=+1} (\vec{w}^*.\vec{x}_i) + \max_{i:y_i=-1} (\vec{w}^*.\vec{x}_i) + 2b^* = 0,$$

and therefore b^* can be recovered from $\vec{w}^*$ and the training set by the formula:

$$b^* = -\frac{1}{2}\left[\min_{i:y_i=+1} (\vec{w}^*.\vec{x}_i) + \max_{i:y_i=-1} (\vec{w}^*.\vec{x}_i)\right].$$

We can now summarize the dual approach to finding the optimal hyperplan in the following proposition:

Proposition 1 *For any separable training set:*

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\},$$

let $\vec{\lambda}^* = (\lambda_1^*, \ldots, \lambda_N^*)$ *be the vector solution of the following constrained optimization problem (the* dual problem*):*

Maximize

$$W(\vec{\lambda}) = -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j + \sum_{i=1}^{N} \lambda_i,$$

under the constraints:

$$\begin{cases} \sum_{i=0}^{N} y_i \lambda_i & = 0, \\ \lambda_i & \geq 0 \quad \text{for } i = 1, \ldots, N. \end{cases}$$

Then the pair $(\vec{w}^*, b^*)$ *defined by:*

$$\begin{cases} \vec{w}^* & = \sum_{i=1}^{N} \lambda_i^* y_i \vec{x}_i, \\ b^* & = -\frac{1}{2}\left[\min_{i:y_i=+1} (\vec{w}^*.\vec{x}_i) + \max_{i:y_i=-1} (\vec{w}^*.\vec{x}_i)\right], \end{cases}$$

defines the optimal hyperplane.

3.6 Support vectors

Let us try to get an intuition of the result obtained in the preceding section. Our goal was to find the optimal hyperplane for a given separable training set. We formulated this problem as a constrained optimization problem (in section 3.4) and explained how to solve it (Proposition 1) using a dual formulation.

In the dual formulation (see Chapter 7) we find a dual vector $\vec{\lambda}^* = (\vec{\lambda}_1^*, \ldots, \vec{\lambda}_N^*)$ and use it to recover the optimal hyperplan $(\vec{w}^*, b^*)$. Each component of the dual vector is associated to one constraint, and we know from the general theorem that there are two kinds of constraints:

- the constraints for which $\lambda_i > 0$ are *active*;
- the constraints for which $\lambda_i = 0$ are *inactive*.

In our case the i-th constraint is:

$$y_i(\vec{w}.\vec{x}_i + b) - 1 \geq 0.$$

As we saw on Figure 3.2 this constraint means that the point $\vec{x}_i$ is on the "good" side of the separating hyperplane, and that its distance from that hyperplane is larger than a given margin. More over this constraint is active (i.e., $y_i(\vec{w}.\vec{x}_i + b) - 1 = 0.$) if and only if the distance is exactly equal to the margin.

The interpretation of active and inactive constraints is therefore illustrated in Figure 3.6: *the only active constraints correspond to the points whose distance to the optimal hyperplan is exactly equal to the margin.*

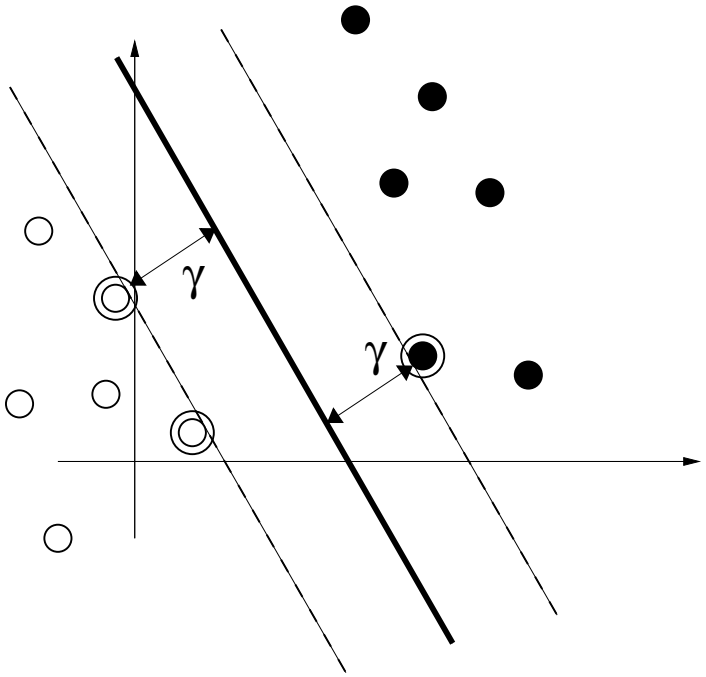

Figure 3.7: The optimal hyperplane, and circles around the support vectors

These particular points (with a circle around them on Figure 3.6) are called *support vectors (abbreviated SV).* They are characterized by the fact that their corresponding constraint is active (i.e., $\lambda_i > 0$), whereas the constraints corresponding to other points are not active (i.e., $\lambda_i = 0$).

In the previous section we showed that the optimal hyperplane $(\vec{w}^*, b^*)$ is characterized by the fact the vector $\vec{w}^*$ is a linear combination of training ex-

amples:

$$\vec{w}^* = \sum_{i=1}^{N} \lambda_i^* y_i \vec{x}_i, \tag{3.11}$$

where $\lambda_i = 0$ if the constraint is inactive, and $\lambda_i > 0$ when the constraint is active. By discarding the null terms in equation 3.11 we can therefore rewrite the optimal vector $\vec{w}^*$ as:

$$\vec{w}^* = \sum_{x_i \text{ is a SV}} \lambda_i y_i \vec{x}_i.$$

As a result $\vec{w}^*$ *is a linear combination of the support vectors.* This has several important consequences:

- The number of support vectors can be very small compared to the size of the training set (in Figure 3.6 there are 3 support vectors out of 12 examples in the training set). Hence $\vec{w}^*$ is a combination of a small number of vectors. This property is called *sparseness*. On a computational point of view special algorithms have been developed to find the vector $\vec{\lambda}^*$ efficiently knowing that most of its components are zero.
- The optimal hyperplane is not influenced by training examples which are not support vectors. In other words, such examples can be removed or added to the training set without any influence on the optimal hyperplane.

3.7 Using SVM for classification

Up to now we have described the SVM algorithm for learning a linear classifier from a training set. The result of the constrained optimization is a pair of vectors $(\vec{w}^*, b^*)$ which defines an optimal hyperplane, and $\vec{w}^*$ can be written as a linear combination of support vectors:

$$\vec{w}^* = \sum_{x_i \text{ is a SV}} \lambda_i^* y_i \vec{x}_i.$$

The goal of SVM learning is precisely to learn the parameters λ_i^* from which the optimal hyperplan can be recovered.

Once learning is finished, how to use a SVM to classify a new observation? Remember that the linear classifier f defined by the vector $(\vec{w}, b)$ classifies any vector $\vec{x}$ according to the following rule:

$$f(\vec{x}) = \begin{cases} +1 \text{ if } \vec{w}.\vec{x} + b > 0, \\ -1 \text{ if } \vec{w}.\vec{x} + b < 0. \end{cases}$$

Hence the classification only depends on the sign of the expression $\vec{w}.\vec{x} + b$. In the case of the SVM classifier $(\vec{w}^*, b^*)$ we can use the property of $\vec{w}^*$ to be

expressed as a linear combination of support vectors to expand this expression as follows:

$$\vec{w}^*.\vec{x} + b = \sum_{x_i \text{ is a SV}} \lambda_i^* y_i \vec{x}_i.\vec{x} + b^*.$$

This formulation gives an intuitive interpretation of the dual variables λ_i^* : the larger λ_i^*, the more important the comparison between $\vec{x}_i$ and $\vec{x}$ (via their dot product) in the final classifier. As an example, points which are not support vectors (i.e., $\lambda_i^* = 0$) have no influence on the classification of a new observation.

This formulation also shows that once the parameters λ_i^* and b^* have been learned, *the classification of a new observation $\vec{x}$ only requires the computation of the dot product between $\vec{x}$ and every support vector.* This fundamental property has the following consequences:

- The number of support vector machines is usually very small compared to the size of the training set, so the classification of a new observation is very quick (it requires a number of operations proportional to the number of support vectors, whatever the size of the training set).
- To classify a new sample one just need to know the values of $\vec{x}.\vec{x}_i$ for each support vector $\vec{x}_i$. This property will be used in Chapter 5 to create non-linear SVM, by replacing this dot product by a so-called *kernel*. The SVM used in real-world applications usually use kernels instead of a single dot product, but we will see in Chapter 5 that the theory of kernel-based SVM is a very simple generalization of linear SVM thanks to this property.

Chapter 4

Linear SVM for general training sets

In Chapter 3 we assumed that the training set $\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ was separable, i.e., that it was possible to find linear classifiers which made no classification error on that set; in that case we showed how to find the one with the largest margin.

In real-life applications this assumption is very often too strong. A general training set might not be linearly separable for at least two reasons:

- There might be noise, measurement errors or outliers in the training set;
- The two classes might not be separated by a single linear classifier, but might require more sophisticated shapes (i.e., non-linear separation rules)

In this chapter we are going to see how it is possible to adapt the linear SVM described in Chapter 3 to handle such training sets. The resulting classifier will still be linear, but classification errors will be accepted in the training set. We will see in Chapter 5 how to build non-linear classifiers.

4.1 Linear classifiers and general training sets

An example of a non-separable training set is shows on Figure 4.1. No linear classifier can correctly classify all examples in that training set. However it is intuitive that some linear classifiers are better than other, even though none of them is perfect. For example on Figure 4.1 the classifier H_1 looks better than the classifier H_2, because it makes fewer mistakes on the training set.

Let us now quantify this intuition. Remember from the last chapter that a pair $(\vec{w}, b)$ defines a "tube", which is the set of points between the hyperplanes $\vec{w}.\vec{x} + b = -1$ and $\vec{w}.\vec{x} + b = +1$. Such a tube makes no mistake on the training set if all positive training point satisfy $\vec{w}.\vec{x} + b \geq +1$ and all negative training points satisfy $\vec{w}.\vec{x} + b \leq -1$.

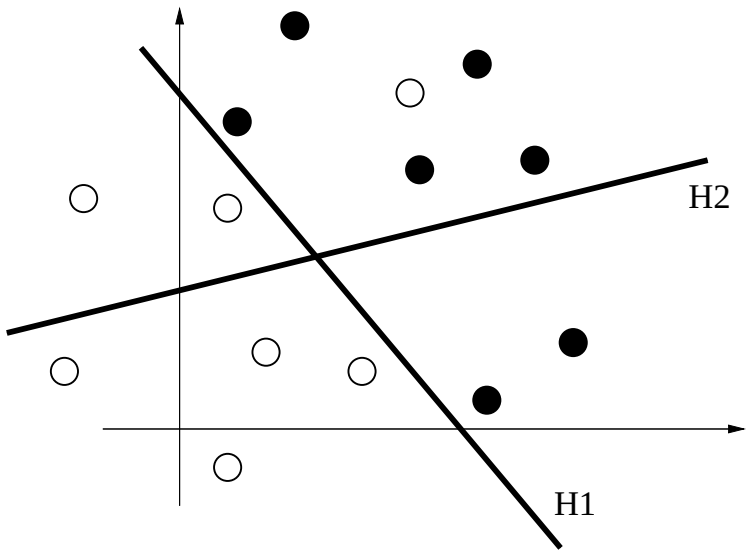

Figure 4.1: Non-separable training set : black circles are positive examples, white circles are negative examples

In order to measure the "amount of mistake" made by a tube of a particular training example, we introduce the following so-called *slack variables*:

- If $y_i = +1$ (positive example) then:

$$\zeta_i(\vec{w}, b) = \begin{cases} 0 & \text{if } \vec{w}.\vec{x} + b \geq +1, \\ 1 - (\vec{w}.\vec{x} + b) & \text{if } \vec{w}.\vec{x} + b \leq +1. \end{cases}$$

- If $y_i = -1$ (negative example) then:

$$\zeta_i(\vec{w}, b) = \begin{cases} 0 & \text{if } \vec{w}.\vec{x} + b \leq -1, \\ 1 + (\vec{w}.\vec{x} + b) & \text{if } \vec{w}.\vec{x} + b \geq -1. \end{cases}$$

In other words the slack variable $\zeta_i(\vec{w}, b)$ corresponding to the example $(\vec{x}_i, y_i)$ is zero if the example is on the "good side" of the tube defined by $\vec{w}$ and b, otherwise it has a positive value which measures the distance between the point x_i and the tube frontier $\vec{w}.\vec{x} + b = y_i$. In particular a training point $(\vec{x}_i, y_i)$ is misclassified by the linear classifier $(\vec{w}, b)$ when $\zeta_i > 1$ (see Figure 4.1). Observe that the definitions of the slack variable for positive and negative examples can be summarized in the following equation:

$$\text{For } i = 1, \ldots, N, \quad \zeta_i(\vec{w}, b) = \begin{cases} 0 & \text{if } y_i\,(\vec{w}.\vec{x} + b) \geq 1, \\ 1 - y_i\,(\vec{w}.\vec{x} + b) & \text{if } y_i\,(\vec{w}.\vec{x} + b) \leq 1. \end{cases} \tag{4.1}$$

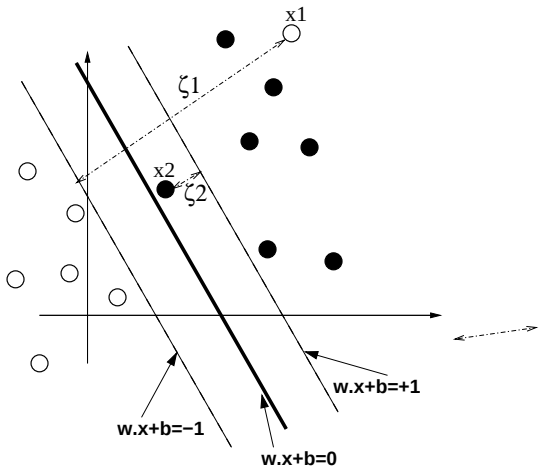

Figure 4.2: Slack variables

For a given training set, what is a "good" linear classifier? Following the discussions in Chapter 2 and 3 a good linear classifier $(\vec{w}, b)$ should have the two following properties:

- The number and the "importance" of mistakes on the training set should be small, i.e., the slack variables $\zeta_i(\vec{w}, b)$ should be as small as possible for $i = 1, \ldots, N$ (in the limit, if they are all equal to zero, then the classifier $(\vec{w}, b)$ makes no mistake on the training set).

- As in Chapter 3 the width of the tube corresponding to the pair $(\vec{w}, b)$ should be as large as possible (to ensure good generalization performances), which means that $||\vec{w}||$ should be as small as possible.

A simple way to combine these two constraints into a single value is the measure the goodness of a linear classifier $(\vec{w}, b)$ by the following number:

$$\epsilon(\vec{w}, b) = ||\vec{w}||^2 + C \sum_{i=1}^{N} \zeta_i(\vec{w}, b), \tag{4.2}$$

where C is a parameter which we can freely choose to tune the trade-off between the width of the tube on the one hand and the number of errors in the training set on the other hand.

This number $\epsilon(\vec{w}, b)$ is a measure of how "good" a linear classifier is on a general training set: *the smaller* $\epsilon(\vec{w}, b)$ *the better the classifier* $(\vec{w}, b)$. As a result a linear SVM on a general training set look for the classifier $(\vec{w}, b)$ with the smallest value of $\epsilon(\vec{w}, b)$, which is nothing but a generalization of the SVM for separable training set, where errors are allowed on the training set.

4.2 Finding the optimal linear classifier

Hence the linear SVM on a general training set solves the following problem:

Find the pair $(\vec{w}, b)$ which minimizes:

$$\epsilon(\vec{w}, b) = ||\vec{w}||^2 + C \sum_{i=1}^{N} \zeta_i(\vec{w}, b).$$

Observe that there is no constraint on $\vec{w}$ and b, so this is simply an minimization problem. It is however not convenient to solve it directly because the functions $\zeta_i(\vec{w}, b)$ are not differentiable as functions of $\vec{w}$ and b, so classical methods involving the computation of the gradient of the function to minimize can not be applied here.

Let us consider instead the following problem:

Find the pair $(\vec{w}, b)$ and the vector $\vec{\xi} = (\xi_1, \ldots, \xi_N)$ which minimize:

$$||\vec{w}||^2 + C \sum_{i=1}^{N} \xi_i$$

under the constraints:

$$\xi_i \geq \zeta_i(\vec{w}, b) \quad \text{for } i = 1, \ldots, N.$$

Obviously the solution of this problem is to take $\xi_i = \zeta_i(\vec{w}, b)$ and to rewrite it with only two variables $\vec{w}, b$, which gives back the first optimization problem. Hence these two problems are equivalent. However it is more interesting to let $\vec{\xi}$ vary and to rewrite the constraint:

$$\xi_i \geq \zeta_i(\vec{w}, b) \tag{4.3}$$

as follows:

$$\begin{cases} \xi_i \geq 0, \\ \xi_i \geq 1 - y_i \left(\vec{w}.\vec{x}_i + b\right). \end{cases} \tag{4.4}$$

By definition of the slack variables in Eq. (4.1) the two conditions (4.3) and (4.4) are equivalent. As a result we can rewrite the preceding minimization problem as follows:

Find the pair $(\vec{w}, b)$ and the vector $\vec{\xi} = (\xi_1, \ldots, \xi_N)$ which minimize:

$$||\vec{w}||^2 + C \sum_{i=1}^{N} \xi_i$$

under the constraints:

$$\begin{cases} \xi_i \geq 0, \\ \xi_i - 1 + y_i \left(\vec{w}.\vec{x}_i + b\right) \geq 0. \end{cases}$$

This problem is to minimize a convex differentiable function under linear constraints, which we know how to solve (see Chapter 7). Let us do it now.

4.3 Solving the optimization problem

In order to solve the constrained optimization problem obtained in the preceding section we follow the general approach explained in Chapter 7.

We need to introduce dual variables $\vec{\lambda} = (\lambda_1, \ldots, \lambda_N)$ for each of the constraints $\xi_i - 1 + y_i\,(\vec{w}.\vec{x}_i + b) \geq 0$, as well as dual variables $\vec{\mu} = (\mu_1, \ldots, \mu_N)$ for each of the constraints $\xi_i \geq 0$. We can then explicit the Lagrangian as follows:

$$L(\vec{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = ||\vec{w}||^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \lambda_i \left[\xi_i - 1 + y_i\,(\vec{w}.\vec{x}_i + b)\right] - \sum_{i=1}^{N} \mu_i \xi_i. \tag{4.5}$$

For each dual vector $(\vec{\lambda}, \vec{\mu})$ we need to minimize the Lagrangian as a function of $(\vec{w}, b, \vec{x})$. This is done by computing the derivatives as follows:

$$\begin{aligned}
&\frac{\partial L}{\partial \vec{w}}(\vec{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = 2\vec{w} - \sum_{i=1}^{N} y_i \lambda_i \vec{x}_i = 0,\\
&\frac{\partial L}{\partial b}(\vec{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = \sum_{i=1}^{N} y_i \lambda_i = 0,\\
&\frac{\partial L}{\partial \xi_i}(\vec{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = C - \lambda_i - \mu_i = 0, \quad \text{for } i = 1, \ldots, N.
\end{aligned}$$

Using these conditions we can follow exactly the same computations as in the separable training set case to get:

$$\inf_{\vec{w}, b, \vec{\xi}} L(\vec{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j + \sum_{i=1}^{N} \lambda_i,$$

if $\sum_{i=1}^{N} y_i \lambda_i = 0$ and $\lambda_i + \mu_i = C$ for $i = 1, \ldots, N$, $-\infty$ otherwise.

Now this function has to be maximized in $\vec{\lambda} \geq 0$ and $\vec{\mu} \geq 0$. But $\vec{\mu}$ does not appear in the function to maximize, so we just need to maximize it as a function of $\vec{\lambda}$ and to check that there exists some $\mu \geq 0$ for which all constraints are satisfied. This will be the case if and only if $\lambda_i \leq C$ for $i = 1, \ldots, N$, because in that case we can find $\mu_i \geq 0$ such that $\mu_i + \lambda_i = C$ is satisfied.

As a result the dual problem becomes:

Find $\vec{\lambda} = (\lambda_1, \ldots, \lambda_N)$ which minimizes

$$W(\vec{\lambda}) = -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j + \sum_{i=1}^{N} \lambda_i,$$

under the constraints:

$$\begin{cases} \sum_{i=0}^{N} y_i \lambda_i & = 0, \\ 0 \leq \lambda_i \leq 0 \quad \text{for } i = 1, \ldots, N. & \end{cases}$$

Once $\vec{\lambda}$ is found one recovers the other dual vector $\vec{\mu}$ thanks to the constraint:

$$\mu_i = C - \lambda_i, \quad \text{for } i = 1, \ldots, N.$$

One can then recover the optimal tube $(\vec{w}^*, b^*)$ thanks to the constraint:

$$\vec{w}^* = \frac{1}{2} \sum_{i=1}^{N} y_i \lambda_i \vec{x}_i,$$

and thanks to the equation:

$$b = -\frac{1}{2} \left[\min_{i:y_i=+1} (\vec{w}^*.\vec{x}_i) + \max_{i:y_i=-1} (\vec{w}^*.\vec{x}_i) \right].$$

(just like in the separable case).

4.4 Comparison with the separable case

The result of the analysis in the previous sections leads to a result which is very close to the result obtained in Chapter 3 when we assumed a separable training set. The "recipe" for the linear SVM with a general training set can be summarized as follows:

Proposition 2 *For any training set:*

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\},$$

let $\vec{\lambda}^* = (\lambda_1^*, \ldots, \lambda_N^*)$ *be the vector solution of the following constrained optimization problem:*

Maximize

$$W(\vec{\lambda}) = -\frac{1}{4} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \vec{x}_i.\vec{x}_j + \sum_{i=1}^{N} \lambda_i,$$

under the constraints:

$$\begin{cases} \sum_{i=0}^{N} y_i \lambda_i & = 0, \\ 0 \leq \lambda_i \leq C \quad \textit{for } i = 1, \ldots, N. \end{cases}$$

Then the pair $(\vec{w}^*, b^*)$ *defined by:*

$$\begin{cases} \vec{w}^* & = \sum_{i=1}^{N} \lambda_i^* y_i \vec{x}_i, \\ b^* & = -\frac{1}{2} \left[\min_{i:y_i=+1} (\vec{w}^*.\vec{x}_i) + \max_{i:y_i=-1} (\vec{w}^*.\vec{x}_i) \right], \end{cases}$$

defines the optimal hyperplane, which minimizes:

$$\epsilon(\vec{w}, b) = ||\vec{w}||^2 + C \sum_{i=1}^{N} \zeta_i(\vec{w}, b).$$

This is very similar to Proposition 1, except that the constraint on $\vec{\lambda}$ is:

$$0 \leq \vec{\lambda} \leq C,$$

instead of:

$$0 \leq \vec{\lambda}.$$

This new constraint is called a "Box" constraint. If the condition $\vec{\lambda} \leq C$ was not used then the maximum of the function $W(\vec{\lambda})$ would be $+\infty$, reached when $\vec{\lambda} = +\infty$, when the training set is not separable. Using this constraint there is always a finite solution to the maximization of $W(\vec{\lambda})$ in the box $0 \leq \vec{\lambda} \leq C$.

4.5 Interpretation for $\vec{\lambda}$ and $\vec{\mu}$

Recall that λ_i is the dual variable associated with the constraint:

$$\xi_i - 1 + y_i \left(\vec{w}.\vec{x}_i + b\right) \geq 0,$$

and μ_i is the dual variable associated with the constraint:

$$\xi_i \geq 0.$$

Moreover, $\lambda_i + \mu_i = C$. As a result the values of μ_i and λ_i depend on the point $(\vec{x}_i, y_i)$ as follows (see Figure 4.5):

- $(\lambda_i = 0, \mu_i = C)$: in that case the constraint corresponding to mu_i is active (because $\mu_i > 0$), which means that $\xi_i = 0$. Hence the point is on the "good" side of the tube, because the slack variable is equal to zero. Moreover, $\lambda_i = 0$ implies that the corresponding constrain is inactive, i.e.,

  $$y_i \left(\vec{w}.\vec{x}_i + b\right) > 1.$$

  In other words, the point is strictly on the good side of the tube, i.e., is not on the frontier.

- $(0 < \lambda_i < C, 0 < \mu_i < C)$: in that case both constraints are active, which means that $\xi_i = 0$ and $y_i \left(\vec{w}.\vec{x}_i + b\right) = 1$. This corresponds to points which are exactly on the frontier of the tube, i.e., which we called support vectors in the preceding chapter.

- $(\lambda_i = C, \mu_i = 0)$: in that case the constraint corresponding to μ_i is inactive, i.e., $\xi_i > 0$. In other words the point is not on the good side of the tube.

4.6 Classifying new examples

As in the separable case, the optimal vector $\vec{w}^*$ is expressed as a linear combination of the training points, namely:

$$\vec{w}^* = \sum_{i=1}^{N} y_i \lambda_i \vec{x}_i.$$

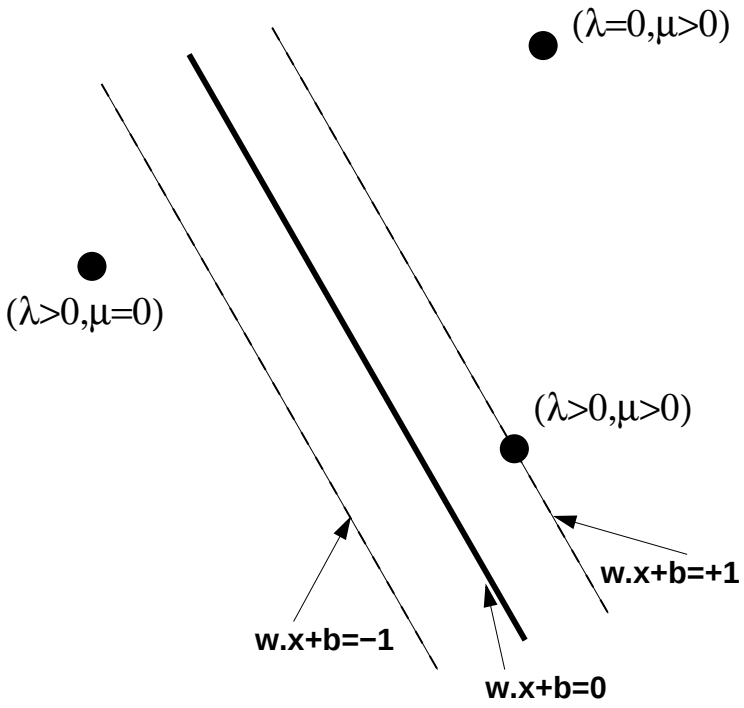

Figure 4.3: Dual variables interpretation

As a result the classification of a new example $\vec{x}$ is based on the sign of the function:

$$f(\vec{x}) = \sum_{i=1}^{N} y_i \lambda_i \vec{x}_i . \vec{x} + b^*, \tag{4.6}$$

where b^* is computed as in Proposition 2. This shows that once again, one does not really need to explicitly compute $\vec{w}^*$ in order to classify new points : once the optimal $\vec{\lambda}^*$ has been computed (by numerically solving the constrained optimization problem), one can directly use the values $\lambda_i, \ldots, \lambda_N$ to classify any new point $\vec{x}$ using Eq. (4.6). This property will be useful in Chapter 5.

Chapter 5

Non-linear SVM : using kernels

In Chapters 3 and 4 we defined linear SVM. The resulting classifier is very simple, because it simply classifies a point as positive or negative depending on whether the point is on one side or on the other side of an hyperplane, e.g., a line in 2 dimensions.

Such classifier are clearly too simple to reflect the complexity of some tasks. As an example, consider the training set represented in Figure 5. Any linear classifier will do a bad job on this training set, which however looks very simple to separate if one could use circles instead of lines. In this chapter we show how SVM can be very easily generalized to handle such cases.

5.1 Feature space

The training set $\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ is a set of labelled example.

Suppose that one is able to define a set of real functions $\phi_1, \ldots, \phi_M$ on the space of objects. This functions are called *features*. Then any object $\vec{x}$ can be mapped to a real vector $\phi(\vec{x})$ with dimension M as follows:

$$\vec{x} = (x_1, \ldots, x_m) \to \phi(\vec{x}) = (\phi_1(\vec{x}), \ldots, \phi_M). \tag{5.1}$$

The features ϕ_i can be any function. In particular they don't need to be linear. Moreover the number of features M can be larger than the dimension m of the objects $\vec{x}$.

After mapping all the points from the training set to the feature space, one gets a set of points:

$$\phi(\mathcal{S}) = \{(\phi(\vec{x}_1), y_1), \ldots, (\phi(\vec{x}_N), y_N)\},$$

in the feature space $\mathbb{R}^M$. The interesting fact about the features space is that *the training set $\phi(\mathcal{S})$ can be linearly separable in the feature set even if the training*

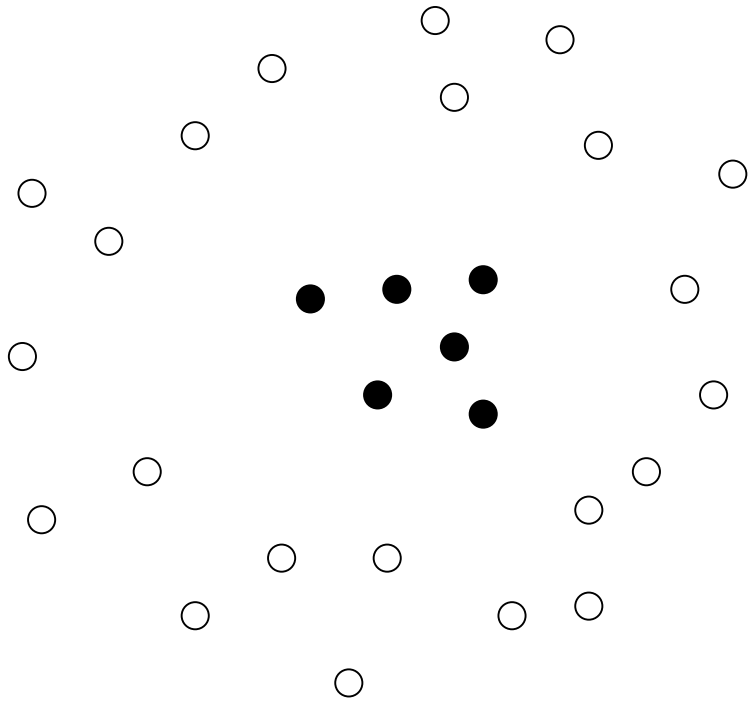

Figure 5.1: Non-separable training set

set is not linearly separable in the original space. An example is illustrated in Figure 5.1

Example 5 *Suppose the initial data are two-dimensional points, i.e., $m = 2$ and $\vec{x} = (x_1, x_2)$. Consider the following mapping:*

$$\vec{x} = (x_1, x_2) \rightarrow \phi(\vec{x}) = (x_1^2, x_1 x_2, x_2^2).$$

Then a general linear hyperplane in the feature space is defined by a vector $\vec{w} = (w_1, w_2, w_3)$ and a number b through the equation:

$$\vec{w}.\phi(\vec{x}) + b = 0.$$

This equation can be explicited as:

$$w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + b = 0.$$

Even though this is a linear equation in the feature space, this corresponds to a polynomial equation in the input space $\mathbb{R}^2$. As a result the set of linear classifier in the feature space is in fact the set of polynomial classifier in the input space. For example, the disk centered in O with radius R is defined by the equation:

$$x_1^2 + x_2^2 < R^2,$$

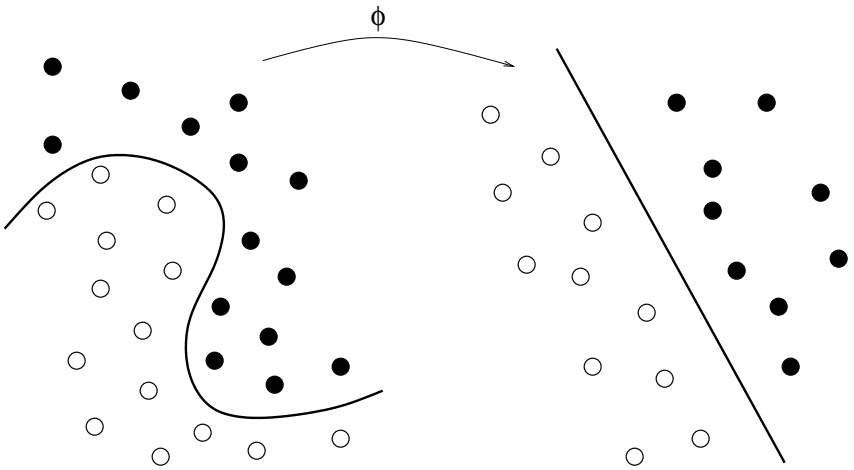

Figure 5.2: Linear separation in the feature space

which corresponds to the vector $\vec{w} = (1, 0, 1)$ *and* $b = -R^2$ *in the feature space. Hence the example shown in Figure 5 can be easily separated by a linear classifier once mapped to this feature space.*

As a result the SVM approach can be generalized to non-linear classification by the following steps:

- Define a mapping ϕ to a feature space;
- Build a linear SVM in the feature space.

5.2 Linear SVM in the feature space

In the feature space the training set is:

$$\phi(\mathcal{S}) = \{(\phi(\vec{x}_1), y_1), \ldots, (\phi(\vec{x}_N), y_N)\}.$$

Following our analysis in Chapter 4, we know (see Proposition 2)that a linear SVM in the feature space computes a dual vector $\vec{\lambda} = (\lambda_1, \ldots, \lambda_N)$ by maximizing the function:

$$W(\vec{\lambda}) = -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j \phi(\vec{x}_i).\phi(\vec{x}_j) + \sum_{i=1}^{N} \lambda_i,$$

under the constraints:

$$\begin{cases} \sum_{i=0}^{N} y_i \lambda_i & = 0, \\ 0 \leq \lambda_i \leq C \quad \text{for } i = 1, \ldots, N. \end{cases}$$

From this $\vec{\lambda}^*$ one can compute the number b^* by the equation:

$$b^* = -\frac{1}{2}\left[\min_{i:y_i=+1}(\vec{w}^*.\vec{x}_i) + \max_{i:y_i=-1}(\vec{w}^*.\vec{x}_i)\right],$$

and the resulting classifier classifies any new example $\vec{x}$ depending on the sign of the function:

$$f(\vec{x}) = \sum_{i=1}^{N} y_i \lambda_i^* \phi(\vec{x}_i).\phi(\vec{x}) + b^*.$$

Observe that we did not explicitly compute $\vec{w}^*$ because we don't need it to define the optimal classifier.

From these equations we see that all we need to know about the mapping ϕ is the value of the dot product $\phi(\vec{x}).\phi(\vec{x}')$ for any two points $(\vec{x}, \vec{x}')$ in the input space. This leads to the following definition:

Definition 1 *A* kernel $K(.,.)$ *is a function such that for any points* (x, x') *in the input space:*

$$K(\vec{x}, \vec{x}') = \phi(\vec{x}).\phi(\vec{x}')$$

where ϕ *is a mapping to a feature space.*

Example 6 *Consider the mapping:*

$$\vec{x} = (x_1, x_2) \rightarrow \phi(\vec{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

Then the corresponding kernel is:

$$\begin{aligned} K(\vec{x}, \vec{x}') &= \phi(\vec{x}).\phi(\vec{x}') \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2).\left((x_1')^2, \sqrt{2}x_1'x_2', (x_2')^2\right) \\ &= x_1^2(x_1')^2 + 2x_1x_2x_1'x_2' + x_2^2(x_2')^2 \\ &= (x_1x_1' + x_2x_2')^2 \\ &= (\vec{x}.\vec{x}')^2 \end{aligned}$$

Using the kernel $K(x, x')$ to replace the dot product $\phi(x).\phi(x')$ in the SVM algorithm we get the following "recipe" for the kernel-based SVM on a general training set:

Proposition 3 *For any training set:*

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\},$$

and any kernel $K(.,.)$ *corresponding to a dot product in a feature space, let* $\vec{\lambda}^* = (\lambda_1^*, \ldots, \lambda_N^*)$ *be the vector solution of the following constrained optimization problem:*

Maximize

$$W(\vec{\lambda}) = -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j K(\vec{x}_i, \vec{x}_j) + \sum_{i=1}^{N} \lambda_i,$$

under the constraints:

$$\begin{cases} \sum_{i=0}^{N} y_i \lambda_i & = 0, \\ 0 \leq \lambda_i \leq C \quad \textit{for } i = 1, \ldots, N. \end{cases}$$

Let the decision function $f(.)$ be defined for any object $\vec{x}$ by:

$$f(x) = \sum_{i=1}^{N} y_i \lambda_i^* K(\vec{x}_i, \vec{x}) + b^*,$$

where b^* is chosen so that $y_i f(\vec{x}_i) = 1$ for any i with $0 < \lambda_i < C$.
Then the classifier which classifies any point $\vec{x}$ as positive if $f(\vec{x}) > 0$ and as negative otherwise is called a kernel-based SVM. It corresponds to a linear SVM in the feature space.

In Proposition 3 we did not explicit the vector $\vec{w}^*$ because the classification function can be defined in terms of $\vec{\lambda}^*$. To compute b^* we just used the fact that if $0 < \lambda_i < C$ then the corresponding point is on the optimal tube limit.

5.3 Implicit mapping to a feature space

A kernel $K(.,.)$ always corresponds to a dot product in a particular feature space defined by a mapping $\phi(.)$. Hence the most natural way to build a kernel is first to define a mapping $\phi(.)$, and then to compute the corresponding kernel (as we did in Example 5).

However the computation of the kernel $K(\vec{x}, \vec{x}')$ can be sometimes much easier than the explicit computation of the mapping $\phi(.)$. In Example 5, the kernel is simply:

$$K(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}')^2,$$

while the mapping is:

$$\phi(\vec{x}, \vec{x}') = (x_1^2, x_1 x_2, x_2^2).$$

In other words, it is possible to directly compute the kernel $K(\vec{x}, \vec{x}')$ between two objects without computing their images $\phi(\vec{x})$ and $\phi(\vec{x}')$.

This property is the main reason why it is very useful to use kernels: they are usually simple to calculate, but can correspond to complex feature spaces. Therefore we talk about *implicit mapping* to a feature space, because the data are mapped to a feature space where their dot product can be computed, while their images are not explicitly computed.

5.4 Popular kernels

Let us now introduce some very popular kernels which are used in most SVM packages. They all assume that the original objects $\vec{x}$ are vectors (we will see in Chapter 6 how kernels can be defined directly form objects such as sequences).

5.4.1 Polynomial kernels

Two general polynomials kernels are defined as:

$$K_{Poly1}(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}')^d,$$

and

$$K_{Poly2}(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}' + c)^d,$$

where d is the degree of the polynom and c is a constant in the second kernel.

We already met the polynomial kernel K_{Poly1} of degree 2 in Example 5, it corresponds to a feature space spanned by all products of 2 variables, i.e., $\left\{x_1^2, x_1x_2, x_2^2\right\}$. It is easy to see that the kernel K_{Poly2} of degree 2 corresponds to a feature space spanned by all products of at most 2 variables, i.e., $\left\{1, x_1, x_2, x_1^2, x_1x_2, x_2^2\right\}$.

More generally the kernel K_{Poly1} corresponds to a feature space spanned by all products of exactly d variables, while the kernel K_{Poly2} corresponds to a feature space spanned by all products of at most d variables.

As a result, the decision function are polynomial decision functions, for example:

$$f(x) = x_1^3 - 2x_1x_3 + 4$$

is a possible decision function for the kernel K_{Poly2} of degree 3. The resulting shapes can become complex as the degree increases.

5.4.2 Radial basis function kernel

This kernel is defined by:

$$K(\vec{x}, \vec{x}') = \exp\left(-\frac{||\vec{x} - \vec{x}'||}{2\sigma^2}\right),$$

where σ is a parameter. It corresponds to a feature space with infinite dimension which can not be completely explicited; hence this is a typical example where the explicit mapping to the feature space can not be computed while the dot product in that space is easy to compute.

The corresponding decision functions have the form:

$$f(\vec{x}) = \sum_{i=1}^{N} y_i\lambda_i^* \exp\left(-\frac{||\vec{x}_i - \vec{x}||}{2\sigma^2}\right)$$

and is therefore a sum of Gaussian centered on the support vectors. Almost any shape can be obtained with this kernel. Observe that the smaller the parameter

σ, the more peaked the Gaussians are around the support vectors, and therefore the more complex the decision boundary can be. The larger σ the smoother the decision boundary.

5.4.3 Sigmoid kernel

The sigmoid kernel is defined by:

$$K(\vec{x}, \vec{x}') = \tanh\left(\kappa \vec{x}.\vec{x}' + \theta\right),$$

where κ and θ are parameters respectively called *gain* and *threshold*. Once again the corresponding feature space can not be easily explicited. The decision function:

$$f(\vec{x}) = \sum_{i=1}^{N} \lambda_i^* y_i \tanh\left(\kappa \vec{x}_i.\vec{x} + \theta\right)$$

is a particular type of two-layer sigmoidal neural network.

5.5 Using SVM

As a conclusion the general approach to use SVM is the following:

- Define your problem as a classification problem;
- Prepare a training set

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\};$$

- Choose a kernel $K(.,.)$
- Compute λ^* using numerical methods to solve the constrained optimization problem (see Proposition 3);
- Classify any new object depending on the sign of the decision function:

$$f(\vec{x}) = \sum_{i=1}^{N} \lambda_i^* y_i K(\vec{x}_i, \vec{x})$$

There is usually no automatic way to choose a kernel and to adjust the corresponding parameters. Therefore one usually has to try different kernels and different parameters and to test their efficiency on his data. A good method is to randomly split the data into a training set and an test set, to train the SVM on the training set and to test it on the test set.

Chapter 6

SVM for bioinformatics : new kernels?

Chapter 7

Annex : Optimization theory

In this annex we show how to solve the problem:

Minimize

$$f(\vec{x})$$

under the constraints:

$$\text{For } i = 1, \ldots, N, \qquad \vec{x}.\vec{a}_i + b_i \geq 0,$$

where $\vec{x} = (x_1, \ldots, x_m)$ is a m-dimensional real vector, $f(\vec{x})$ is a smooth strictly convex function of $\vec{x}$ (by smooth we mean that f has continuous derivatives), and for each $i = 1, \ldots, N$, $\vec{a}_1$ is a m-dimensional real vector and b_i is a real number.

7.1 Minimization with no constraint

Let us first consider the case when there is no constraint. Then the problem of interest becomes:

Minimize

$$f(\vec{x}).$$

f being convex, it is well known that a point $\vec{x}^* = (x_1^*, \ldots, x_m^*)$ is a solution of this problem if and only if the derivatives of f with respect to each coordinate of $\vec{x}$ are equal to zero. Let us use the notation:

$$\frac{\partial f}{\partial x_i}(\vec{x})$$

to denote the derivatives of the function f with respect to the i-th coordinate (for $i = 1, \ldots, m$), and $\nabla f(\vec{x})$ to denote the *gradient* of f at the position $\vec{x}$, i.e. the vector whose coordinates are the partial derivatives of f:

$$\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}(\vec{x}), \ldots, \frac{\partial f}{\partial x_m}(\vec{x}) \right).$$

With these notations, we can characterize the solution of the minimization problem as follows:

Theorem 1 *A point $\vec{x}^*$ is a minimum of the smooth convex function $f(\vec{x})$ if and only if $\nabla f(\vec{x}^*) = 0$*

This theorem gives a method to compute the minimum of f: compute $\nabla f(\vec{x})$ and solve the equation $\nabla f(x) = 0$.

Example 7 *To find the minimum of the function:*

$$f(\vec{x}) = f(x_1, x_2) = x_1^2 - 2x_1 + x_2^2,$$

it suffices to compute the partial derivatives of f:

$$\begin{cases} \frac{\partial f}{\partial x_1}(\vec{x}) & = 2x_1 - 2, \\ \frac{\partial f}{\partial x_2}(\vec{x}) & = 2x_2. \end{cases}$$

Hence the gradient vector is null for $\vec{x}^* = (1, 0)$, which is the solution of the minimization problem. The minimum value of f is therefore $f(1, 0) = -1$.

In one dimension ($m = 1$), $\vec{x} = (x_1)$ is simply a real number, f is a function and $\nabla f(\vec{x})$ is the classical derivative of f. In larger dimension, it is useful to keep in mind the picture represented in Figure 7.1, which shows the level sets of f as well as the gradient vector ∇f (a level set of f is a set of points $\vec{x}$ having the same value $f(\vec{x})$).

This picture shows in particular that, at any point $\vec{x}$, the gradient $\nabla f(\vec{x})$ has the following properties:

- it goes to the direction of steepest ascent of the function f from the point $\vec{x}$;
- it is perpendicular to the tangent of the level set of f at the point $\vec{x}$.

7.2 Minimization with one constraint

Let us now consider the minimization problem with only one constraint (N=1). It can be rewritten as:

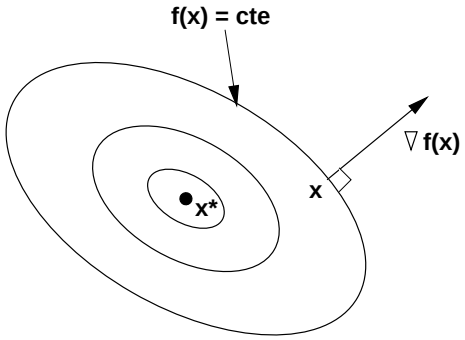

Figure 7.1: Level sets and gradient of a convex function f

Minimize

$$f(\vec{x})$$

under the constraint:

$$\vec{x}.\vec{a} + b \geq 0,$$

where $\vec{a}$ is a m-dimensional vector and b is a real number.

The constraint is linear. Therefore, the set of vectors $\vec{x}$ which satisfy the constraint $\vec{x}.\vec{a} + b \geq 0$ is a half-space delimited by the hyperplane $\vec{x}.\vec{a} + b = 0$, which we call the *allowed half-space*. The set of points for which the constraint is not fulfilled is the other half-space delimited by the same hyperplane, which we call the *forbidden half-space* (see Figure 7.2). This hyperplane $\vec{x}.\vec{a} + b = 0$ is perpendicular to $\vec{a}$, and the allowed half-space is the half-space which is pointed out by the vector $\vec{a}_1$.

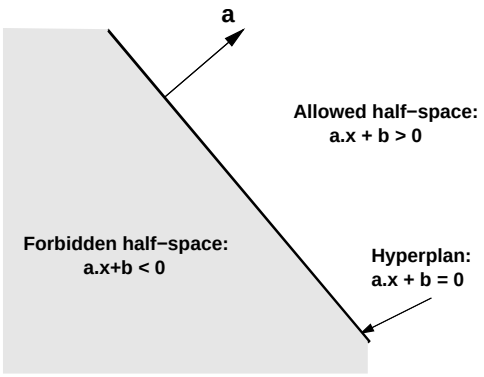

Figure 7.2: Representation of a linear constraint $\vec{a}.\vec{x} + b \geq 0$

Let us call $\vec{x}^*$ the point which minimizes $f(\vec{x})$ under the constraint $\vec{a}.\vec{x} + b \geq 0$, and $\vec{x}_0^*$ the point which minimizes $f(\vec{x})$ without any constraint. Two different

situations might happen, depending on the position of the global minimum $\vec{x}_0^*$:

1. If $\vec{x}_0^*$ belongs to the allowed half-space (see Figure 1), then it is obviously the minimum of f on this half-space, and therefore $\vec{x}^* = \vec{x}_0^*$. In that case $\vec{x}^*$ satisfies:

$$\begin{cases} \nabla f(\vec{x}^*) & = 0, \\ \vec{x}^*.\vec{a} + b & \geq 0. \end{cases} \tag{7.1}$$

   (the first equation characterize the global minimum of f, the second states that it is in the allowed half space). Conversely, if a point $\vec{x}^*$ satisfies the system of equation 7.1 then it is a global minimum (from the first equation) which satisfies the constraint (from the second equation): it is therefore the minimum of the optimization problem. As a result, the system of equations (7.1) characterizes $\vec{x}^*$ in that case.

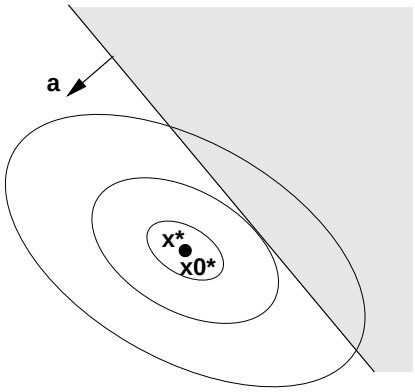

Figure 7.3: Minimization when the global minimum is on the allowed half-space

2. If $\vec{x}_0^*$ is in the forbidden half-space, then it is not the minimum of the constrained problem because it violated the constraint: $\vec{x}^* \neq \vec{x}_0^*$. This situation is depicted in Figure 2. It can be seen on that figure (and it can be mathematically proven) that the point $\vec{x}^*$ has the following properties:

   - $\vec{x}^*$ *is on the frontier hyperplan.* Indeed, if it was not on the frontier then f could be decreased by moving from $\vec{x}^*$ to the opposite direction of $\nabla f(\vec{x}^*)$, while still remaining in the allowed half-space (for example, $f(x_1) < f(x_2)$ on Figure 2).
   - *The level set of* f *at the point* $\vec{x}^*$ *is tangent to the hyperplan.* Indeed, if it was not then f could be decreased by moving on the hyperplan (for example, $f(\vec{x}^*) < f(\vec{x}_1)$ on Figure 2).

   Recall that the gradient $\nabla f(\vec{x})$ is always perpendicular to the level set of f at any point $\vec{x}$ (see Figure 7.1), and that $\vec{a}$ is perpendicular to the

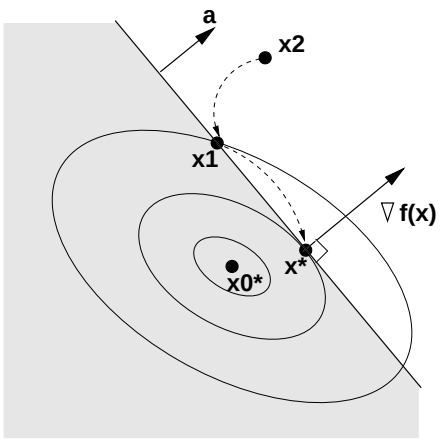

Figure 7.4: Minimization when the global minimum is on the forbidden half-space

frontier hyperplane (Figure 7.2). Therefore the level set is tangent to the frontier on a point $\vec{x}$ if and only if $\nabla f(\vec{x})$ *is parallel to* $\vec{a}_1$. Besides, $\nabla f(\vec{x})$ and a_1 should have the same direction to be sure that the minimum of f is on the forbidden half-space. As a result, the solution of the constrained optimization problem in that case is characterized by:

$$\begin{cases} \nabla f(\vec{x}) = \lambda_1 \vec{a}_1 & \text{for some positive real number } \lambda_1; \\ \vec{x}.\vec{a} + b = 0. \end{cases} \tag{7.2}$$

(the first equation translates the fact that the level set of f should be tangent to the frontier and that the global minimum of f should be in the forbidden space, while the second equation translates the fact that the solution is the frontier). Note that this system of equation is also sufficient : for a vector $\vec{x}$, if there exists a positive real number such that these two equations are satisfied, then $\vec{x}$ is the solution of the constrained optimization problem.

Hence we have characterized the solution of the constrained optimization problem by two possible set of equations: a point $\vec{x}^*$ is the solution of the constrained optimization problem if and only if it satisfies the system (7.1) or it satisfies the system (7.2). It is possible to summarize both cases into one single system of equations as follows:

Theorem 2 *$\vec{x}^*$ is a solution of the constrained optimization problem:*

Minimize

$$f(\vec{x})$$

under the constraint:

$$\vec{x}.\vec{a} + b \geq 0,$$

if and only if there exists a real number λ^* *such that the pair* $(\vec{x}^*, \lambda^*)$ *satisfies:*

$$\begin{cases} \nabla f(\vec{x}^*) = \lambda \vec{a} \\ \vec{x}^*.\vec{a} + b \geq 0 \\ \lambda \geq 0 \\ \lambda.\,(\vec{x}^*.\vec{a} + b) = 0 \end{cases}$$

This theorem is a particular case of the so-called Kuhn-Tucker theorem. We will see in section 7.4 how it can be used to solve the optimization problem, but let us first generalize it to the case when there are several constraints.

7.3 Optimization with several constraint

Let us now consider the constrained optimization problem with several constraints:

Minimize

$$f(\vec{x})$$

under the constraints:

$$\text{For } i = 1, \ldots, N, \qquad \vec{x}.\vec{a}_i + b_i \geq 0,$$

Each constraint defines an allowed and a forbidden half-space separated by the hyperplanes $\vec{x}.\vec{a}_i + b_i = 0$. The intersection of the allowed half-spaces is the set of vector $\vec{x}$ which satisfy all constraints, and the problem is to find a point $\vec{x}^*$ which minimizes f on that set (see Figure 7.3).

It can be shown that the analysis we did with only one constraint in the preceding section can be generalized to the multi-constraint case. Let us just give here the result of this analysis:

- if the global minimum $\vec{x}_0^*$ of f satisfies all constraints then it is the solution of the constrained optimization problem: $\vec{x}^* = \vec{x}_0^*$
- otherwise $\vec{x}^*$ lies on the frontier of the allowed space (see Figure 7.3). It can be at the intersection of several hyperplanes, whose corresponding constraints are said to be *active*. The other constraints are called *inactive*. Moreover, the gradient of f satisfies:

  $$\nabla f(\vec{x}) = \sum \lambda_i \vec{a}_i,$$

  where the sum is over the active constraints and the λ_i's are positive real number.

As in the preceding section the different cases can be summarized into one single statement which characterizes the solution of the constrained optimization problem:

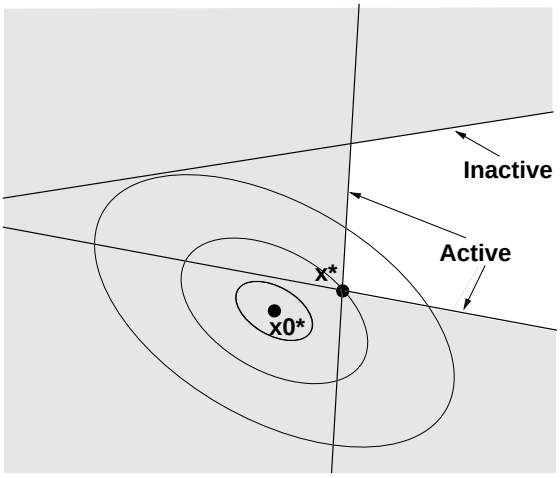

Figure 7.5: Minimization under several linear constraints

Theorem 3 (Kuhn-Tucker) *$\vec{x}^*$ is a solution of the constrained optimization problem:*

Minimize

$$f(\vec{x})$$

under the constraint:

$$\vec{x}.\vec{a}_i + b_i \geq 0 \text{ for } i = 1, \ldots, N$$

if and only if there exists a set of real numbers $\lambda_1^*, \ldots, \lambda_N^*$ such that:

$$\begin{cases} \nabla f(\vec{x}^*) = \lambda_1^* \vec{a}_1 + \ldots + \lambda_N^* \vec{a}_N & \\ \vec{x}^*.\vec{a}_i + b_i \geq 0 & \text{for } i = 1, \ldots, N \\ \lambda_i^* \geq 0 & \text{for } i = 1, \ldots, N \\ \lambda_i^*.(\vec{x}^*.\vec{a}_i + b_i) = 0 & \text{for } i = 1, \ldots, N \end{cases}$$

7.4 Characterizing $(\vec{x}^*, \vec{\lambda}^*)$

Let us denote by $\vec{\lambda}$ the N-dimensional vector:

$$\vec{\lambda} = (\lambda_1, \ldots, \lambda_N).$$

Observe that the size of $\vec{\lambda}$ is N, the number of constraints, while the size of $\vec{x}$ is m, the dimension of the observations. Theorem 3 characterizes the solution $\vec{x}^*$ of the constrained optimization problem by the existence of a vector $\vec{\lambda}$ such

that a set of conditions is fulfilled. Let us now show how a pair $(\vec{x}^*, \vec{\lambda}^*)$ which satisfies these conditions can be found (in that case, $\vec{x}^*$ is the solution of the constrained optimization problem we are looking for).

For that purpose let us introduce a function $L(\vec{x}, \vec{\lambda})$ called the *Lagrangian*, defined by:

$$L(\vec{x}, \vec{\lambda}) = f(\vec{x}) - \sum_{i=1}^{N} \lambda_i \left(\vec{x}.\vec{a}_i + b_i\right). \tag{7.3}$$

Hence L is a function of $N + m$ real numbers. The reason why we introduce this function is that the pair $(\vec{x}^*, \vec{\lambda}^*)$ which satisfies the conditions of Theorem 3 is a very particular point for this function, namely a *saddle point*. Let us explain this by studying some properties of the Lagrangian:

1. For any fixed value of λ, the function $\vec{x} \to L(\vec{x}, \vec{\lambda})$ is convex (because f is convex and remains convex when a linear function is subtracted). As a result there exists at most one minimum when $\vec{x}$ varies and $\vec{\lambda}$ stays fix. Let us call $\vec{x}_{\vec{\lambda}}$ this minimum if it exists (here the minimization is considered without any constraint on $\vec{x}$).

2. For any point $\vec{x}$ which satisfies all the constraints and for any vector $\vec{\lambda}$ with positive values,
$$L(\vec{x}, \vec{\lambda}) \leq f(\vec{x})$$
(because each term $\vec{\lambda}_i.\left(\vec{x}.\vec{a}_i + b_i\right)$ is positive in that case). In particular, for the point $\vec{x}^*$ which minimizes f under the constraints, it is true that for all positive vector $\vec{\lambda}$:
$$L(\vec{x}^*, \vec{\lambda}) \leq f(\vec{x}^*)$$
.

3. As a consequence , for any fixed positive $\vec{\lambda}$, the minimum of the function $\vec{x} \to L(\vec{x}, \vec{\lambda})$ (which we called $L(\vec{x}_{\vec{\lambda}}, \vec{\lambda})$ in the first point) is also smaller than $f(\vec{x}^*)$:
$$L(\vec{x}_{\vec{\lambda}}, \vec{\lambda}) \leq f(\vec{x}^*) \quad \text{for any positive } \vec{\lambda}.$$

4. Consider the particular positive vector $\vec{\lambda}^*$ which satisfies the conditions of Theorem 3 together with the point $\vec{x}^*$. What is the corresponding point $\vec{x}_{\vec{\lambda}^*}$ which minimizes the function $\vec{x} \to L(\vec{x}, \vec{\lambda}^*)$? In order to find the minimum of this convex function it suffices to find the point for which the gradient of $L(., \vec{\lambda})$ (seen as a function of $\vec{x}$) is null. This gradient can be computed from the definition (7.3)of $L(\vec{x}, \vec{\lambda})$ as follows:
$$\text{For any } \vec{x}, \quad \nabla L(\vec{x}, \vec{\lambda}^*) = \nabla f(\vec{x}^*) - \sum_{i=0}^{N} \lambda_i^* \vec{a}_i.$$

If we compute this gradient at the particular point $\vec{x} = \vec{x}^*$, then we can use the fact that:

$$\nabla f(\vec{x}^*) = \sum_{i=1}^{N} \lambda_i \vec{a}_i,$$

which is one of the conditions expressed in Theorem 3. As a result the gradient is null at the point $\vec{x}^*$, which shows that *the minimum of the function* $\vec{x} \to L(\vec{x}, \vec{\lambda}^*)$ *is reached at the point* $\vec{x} = \vec{x}^*$. In other words, $\vec{x}^* = \vec{x}_{\vec{\lambda}^*}$.

5. $(\vec{x}^*, \vec{\lambda}^*)$ satisfy the conditions of Theorem 3. In particular, $\lambda_i \left(\vec{x}.\vec{a}_i + b_i\right) = 0$ for $i = 1, \ldots, N$. As a result, using equation (7.3) we obtain:

$$L(\vec{x}^*, \vec{\lambda}^*) = f(\vec{x}^*).$$

From these observations we can deduce that the point $(\vec{x}^*, \vec{\lambda}^*)$ is a *saddle point* of the Lagrangian $L(\vec{x}, \vec{\lambda})$, in the sense that:

- From point 4 we see that the function $\vec{x} \to L(\vec{x}, \vec{\lambda}^*)$ has a *minimum* at the point $\vec{x} = \vec{x}^*$
- Combining points 2 and 5 we see that the function $\lambda \to L(\vec{x}^*, \vec{\lambda})$ has a *maximum* at the point $\vec{\lambda} = \vec{\lambda}^*$.

This situation is shown on Figure 7.4.

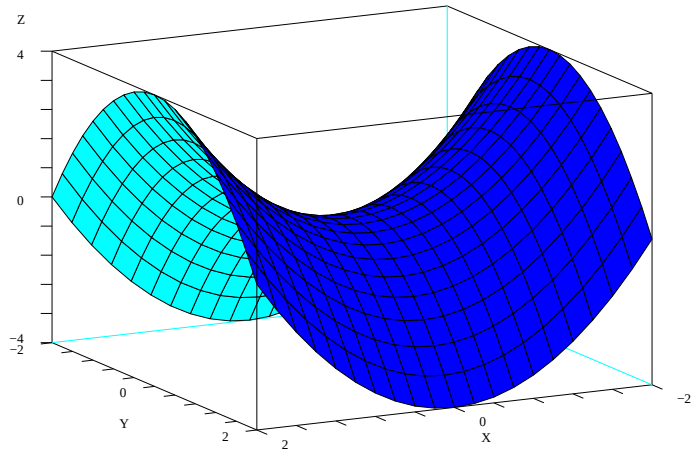

Figure 7.6: A curve with a saddle point

Finally this property can be used to compute $(\vec{x}^*, \vec{\lambda}^*)$ as follows:

1. For any positive λ, compute the vector $\vec{x}_\lambda$ which minimizes the function $\vec{x} \to L(\vec{x}, \vec{\lambda})$ (with no constraint). As there is no constraint on $\vec{x}$ this can easily be done by computing the gradient of this function and making it equal to zero.

2. Consider now the function $\vec{\lambda} \to L(\vec{x}_l, \vec{\lambda})$. The vector $\vec{\lambda}^*$ is the positive vector for which this function is maximal.

3. Once $\vec{\lambda}^*$ is found, $\vec{x}^*$ is recovered thanks to the fact that $\vec{x}^* = \vec{x}_{\vec{\lambda}^*}$.

Steps 1 and 3 can usually be solved analytically, i.e. by direct computation. Step 2, however, requires maximizing a function (of $\vec{\lambda}$) under a set of constraints ($\vec{\lambda}$ has to be positive). In other words this method proposes to replace the original constrained minimization problem by an other constrained maximization problem (step 2 above) which is called the *dual problem*. This problem differs from the original problem (also called the *primal problem*) by several features:

- The maximization is over the vectors $\vec{\lambda}$ which have dimension N (whereas the original minimization problem is over the vectors $\vec{x}$ which have dimension m). Hence this new formulation is especially interesting when $N < m$, i.e. when there are few constraints in a high-dimensional space.

- The constraints on $\vec{\lambda}$ are just that each coordinate has to be positive. This is usually easier to handle than the original constraints on $\vec{x}$.