A COMPUTATIONAL ANALYSIS OF OPTIMIZATION MODELS FOR SUPPORT VECTOR MACHINES

LUCIA PETTERLE

SUPERVISOR: Chiar.mo Prof. Matteo Fischetti

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

ANNO ACCADEMICO 2012/2013

A mia nonna Sergia.

Al mio papi e alla mia mami.

Abstract

The classification problem, known in the statistic field also as discriminant analysis or recognition, is a theme of particular importance for problems resolution in the real world. Possible examples are the medical diagnosis of the benign or malignant nature of a disease, or the pattern image recognition.

As it is well known in the scientific literature for nearly two decades, a valid approach to face the binary classification problem is the one represented by Support Vector Machines. SVMs solve the learning problem by constructing a system which, based on a subset of pre-categorized data and relative features, is able to classify with high accuracy new, not already classified, data. From the mathematical point of view, the SVM approach implies the resolution of a quadratic programming model.

The present thesis work consisted in a computational analysis, conducted via a proper software implementation, of the mathematic behaviour of the aforesaid optimization model (equipped with the so-called Gaussian kernel); in a comparative evaluation of the efficacy of approaches based on the resolution of alternative mathematical models (having in common the intent of reducing overfitting phenomenon); and in a statistical evaluation of the accuracy superiority of a SVM approach classification towards the use of a simpler but much less computationally expensive classifier.

The results we got revealed the intrinsic good nature of Gaussian kernel to be a proper classifier by itself, a characteristic that limits the possible rooms for improvement of accuracy results. We also found that SVM approach, with Gaussian kernel, can boast just a 6% accuracy superiority over a simpler classification approach, that does not require the computationally expensive resolution of a quadratic programming optimization model. This observation, together with an analogous comparative evaluation of the accuracy superiority of SVM approach using linear kernel, enabled us to confirm the aforesaid thesis about Gaussian kernel nature. The previously collected results allowed us to understand that the nature of the linear kernel could offer more degrees of improvement and very preliminary tests seemed to confirm the validity of our alternative approaches. Eventually, the whole procedure produced lower accuracies with respect to the SVM approach, so we leave this kind of approach as a hint for further researches in this area.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis scope

The context of the present work belongs to that discipline of the Artificial Intelligence known in the scientific literature as machine learning. And, specifically, it concerns the so-called classification problem. The classification problem, starting from a historical set of observations (or points or items), each constituted by a set of elements values, named features (or attributes), and a relative category, named class, consists in the assignment of new experimental items to the class considered to be the most correct. The theme of classification has lots of variants and the one we are interested in is classification with just two possible classes for each observation: binary classification. We will concentrate on the approach proposed by Vladimir Vapnik and his colleagues in the 90s, called Support Vector Machines approach.

1.2 Work focus

Our work focuses on the mathematical background that constitutes the algorithmic bases of the functionality of Support Vector Machines. Their operational procedure, that will be described in detail in Chapter 3, refers to the resolution of a quadratic programming problem, whose formulation is strictly related to the choice of the so-called kernel function. It is the key idea on which is based the use of Support Vector Machines as linear classifiers: as a matter of fact, it permits to transform the original features space, where classification can become very arduous, in an equivalent space, where classification can be realized through a simple linear classifier.

First of all in our analysis, that required the implementation of the necessary software and the utilization of appropriated tools for mathematical optimization and statistical evaluations, we investigated in details the behaviour and the role of the Gaussian kernel in the classification procedure.

Secondly, analyzing the values assumed by the classification parameters after the resolution of the aforesaid quadratic programming problem, we made the hypothesis that this approach could have the collateral effect of taking the system to an overfitting behaviour. Therefore, in the aim of reducing the impact of this phenomenon, we planned, implemented and tested alternative mathematical models.

Afterward we shifted our focus to the analysis of the behaviour of the linear kernel. We compared the performance of the system equipped with the classical quadratic model, using linear kernel, with an alternative approach, based on the resolution of a mixed-integer programming problem, aimed again at the reduction of the overfitting problem.

Eventually, we treated an evaluation of the superiority, in terms of accuracy, of the approach adopted by Support Vector Machines that, as previously said, implies the resolution of a quadratic programming problem, towards a much more simplified approach, based on the offline setting of the classification parameters, which can be implemented with an exiguous number of code lines and that requires a very small computational effort.

1.3 Contents structure

In the following we introduce briefly the global structure and the chapter contents of this thesis.

In Chapter 2 we face a panoramic introduction, from the theoretical point of view, of the theme of machine learning and the problem of classification, pointing out the specific aspects on which our focus will be based.

In Chapter 3 we get a deeper sight in a detailed description, from the mathematical point of view, of the approach adopted by Support Vector Machines.

In Chapter 4 we present the alternative approaches we propose to the original SVM approach. We describe the mathematical models tested, motivating the aim of their formulation.

In Chapter 5 we describe the practical effort and the experimental work performed in this thesis, motivating the choices also on theoretical basis, and we provide and analyse the collected experimental results.

In Chapter 6 we treat the characteristics and the modality of use of the software we implemented to perform the experimental tests, in addition to the software that was necessary to implement to interface properly to the first one, and to guarantee an adequate automatization of tests procedures.

In the Appendices we report the experimental results at a larger level of detail, an exemplificative C source code listing and some scripts that were necessary to implement in order to automatize the whole experimental procedures.

Chapter 2

A short introduction to the Classification Problem

2.1 Areas of interest

First of all let us identify the proper areas related to the classification problem.

There are several contexts in applications where it is very important to construct a system which is able to learn how to classify correctly a set of items, using the most significative parameters that characterize the items themselves.

Examples of those contexts are:

The medical field: a system that, based on a sufficient set of past clinical data, is able to discover if a patient is affected by a specific disease (those kinds of systems are really useful in the diagnosis of cancer – see figure 2.1);

The Information Retrieval field: a system that is able to decide if a text is relevant for a specific topic, based on the terms that appear in it;

The image classification field: a system that, based on a small set of points of a simple drawn, is able to reconstruct the pattern below it;

The economical field: a system that is able to determinate different typologies of clients (to produce targeted advertising campaigns).

2.2 Description of the problem

Classification problem is

> *A process for grouping a set of data into groups so that data within a group have high similarity and, at the same time, quite high dissimilarity from data in other groups [2].*

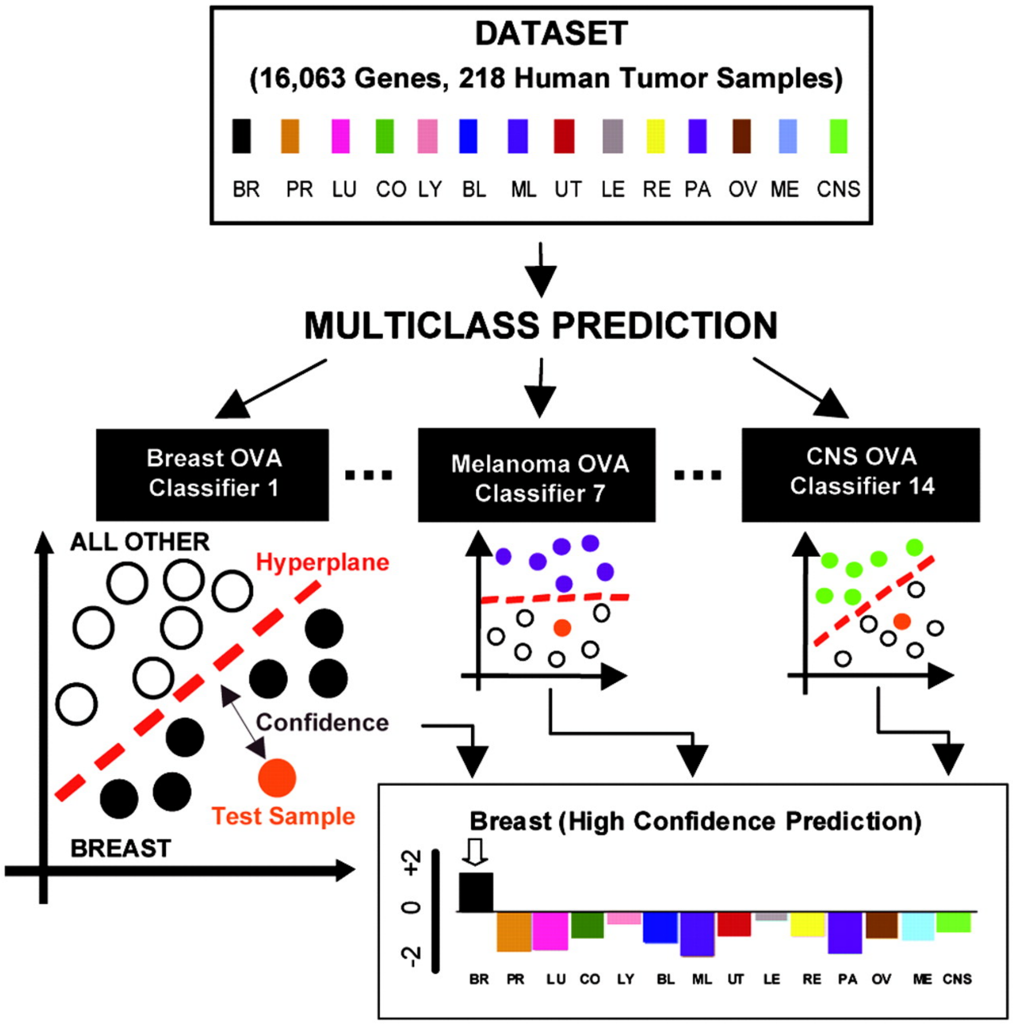

DATASET
(16,063 Genes, 218 Human Tumor Samples)
BR PR LU CO LY BL ML UT LE RE PA OV ME CNS
MULTICLASS PREDICTION
Breast OVA Classifier 1
Melanoma OVA Classifier 7
CNS OVA Classifier 14
ALL OTHER
Hyperplane
Confidence
Test Sample
BREAST
Breast (High Confidence Prediction)
+2
0
-2
BR PR LU CO LY BL ML UT LE RE PA OV ME CNS

Figure 2.1: Multiclass cancer classification scheme.

Classification technique is the most commonly used technique to analyze large sets of data in automatic or semi-automatic way, with the purpose of extracting knowledge from them.

The ascription of a single item to a specific class is the result of a procedure known as *learning procedure*. Typically, a learning procedure is realized by a *learning machine* that, observing an already classified training set of data, constructs an operator that is able to predict the output y for each input x not belonging to the aforesaid training set.

2.2.1 Discriminations in theme of classification

We can make several distinctions in theme of classification.

Multi class *vs* binary classification

There are different problems of classification:

- With several possible classes the single item can belong to: *multi class classification*;
- With only two possible classes the single item can belong to: *binary classification.*

For simplicity we consider only problems of the second type, because they are the problems of interest for our SVM-oriented theoretical overview.

Supervised *vs* unsupervised classification

This is perhaps the most important distinction in classification problems:

- In *supervised* classification we know the possible groups and we have data already classified, being used overall for training. The problem consists in associating the data in the most appropriate group taking advantage of those already labeled.
- In *unsupervised* classification, also-called *clustering*, possible groups (or clusters) are not known in advance, and the data available are not classified. The goal is then to classify in the same cluster the data considered as similar.

We will consider only supervised classification, that is the one used by the SVM approach.

Generative *vs* discriminative classification methods

We can distinguish two different approaches:

- *Discriminative methods:* building a boundary between the classes, they directly model the posterior probability $P(Z|X)$; the empirical observations are explained by the model that describes probabilistically the interaction between the variables of the problem;
- *Generative methods:* deducing the a posteriori probability through the Bayes' Rule, they first model the joint probability distribution P(X,Z); they deal directly with the problem of finding the criteria that permit to group together the empirical observations.

The first approach is the one used by Support Vector Machines.

2.3 Classifier

We refer to the algorithm that realizes classification as *classification function* or *classifier.*

The purpose, in the determination of the classifier, is to minimize the *classification error*: it occurs when an item x is assigned to a class C_i, but actually it belongs to another class C_j. We will talk about different kinds of errors to be minimized in 2.5.1.2.

2.3.1 Steps of the classification problem

In the solution of classification problem we can recognize three fundamental phases:

- construction of a model that describes a certain set of classes after the analysis of some of the multi-dimensional items, through their features: this phase is also-called ***learning phase***;
- evaluation of the constructed model on other items;
- use of the aforesaid model to classify new items: this phase is also-called ***classification phase***.

Given a dataset of points and relative classification, two particular subsets, always strictly disjoint, are taken from the original dataset to implement the learning procedure:

Training set: it permits to *train* the system, so that it is possible to find the most appropriate classification function (it is used during the *learning phase*);

Test set: it permits to realize an estimation of the accuracy of the classification function previously defined (it is used during the *classification phase*).

Some classification methods further divide the training set in k disjoined subsets to perform the so-called *k-fold cross validation.*

2.3.2 K-fold cross validation

Classification techniques often involve the use of special parameters, that can either be defined "offline" or chosen, through a trial-and-error procedure, by the algorithm. Choosing the second possibility, they can be determined before the final training procedure, during the so-called ***validation phase***, in the most proper way in order to give the best classification results.

One of the possible ways to choose the optimal parameters setting is to perform the so-called *k-fold cross validation.*

Cross-validation is a computer intensive technique, introduced in [4]. It mimics the use of training and test sets by repeatedly training the algorithm k times with a fraction $\frac{1}{k}$ of training examples left out for testing purposes.

Analytically, it consists in:

- extracting $\frac{n \cdot (k-1)}{k}$ items from the training set (where n is its dimension);
- constructing a classification rule over the extracted items;
- classifying the remaining $\frac{n}{k}$ items obtaining the related accuracy;
- re-start again from point 1 the whole procedure, k times.

In our experimentation we used 5-fold cross validation that consists in randomly dividing the training set into 5 disjoined subsets: in turn, 4 of the 5 aforesaid subsets assume the role of *training set* and the remaining subset the role of the so-called *validation set.*

Training method is now applied to the previously described training set, and the classification phase takes place over the validation set. This procedure is repeated 5 times, in turn changing the training-validation role by the different fractions of the original training set.

After the completion of the 5 phases, the mean value of the accuracies obtained for each parameter setting is computed: the best mean accuracy obtained identifies the optimal parameter setting to be used during the subsequent learning and classification phases. After validation, the learning procedure is applied over the whole training plus validation set, while classification will take place over the test set.

As terminology can be confusing, let us recap the concepts just explained: validation technique involves to select randomly a fraction of $\frac{4}{5}$ of the training set and to call again this portion "training set" during the validation phase, while "validation set" refers to the complementary portion acting as a fictitious test set.

2.3.3 Leave-one-out

This is a validation technique alternative to the previously described K-fold cross validation [6].

The only difference between the two approaches is that leave-one-out extracts just a single item to be excluded from the training set use during the validation phase. That is, the validation set in each iteration is composed by a single element.

Leave-one-out is more computationally expensive than K-fold cross validation because the aforesaid procedure has to be repeated n times (where n is the dimension of the training set), since each single item of the training set has to be classified during the validation.

Scientific experimentations affirm that the efficacy of leave-one-out can be compared to that of K-fold cross validation [25].

2.4 Mathematical formalization of the problem

Considering just *binary classification* (classification problem in presence of only two possible alternatives), we can formalize the problem as in the following:

> ***Formalization of the classification problem***
> Let X be a generic set (usually $X = \mathbb{R}^n$).
> Let $T = \{(\mathbf{x}_i, y_i), i = 1, \ldots, m\} \subset X \times \{+1, -1\}$ be a set of couples that we call *training set.*
> We want to determinate a map $\delta : X \longrightarrow \{+1, -1\}$, that implements the right association between an item $\mathbf{x}$ that does not belong to the training set and the class y it is associated to with highest probability.

So, classification aims at building a *decision rule* δ:

$$\delta : \mathbb{X} \longrightarrow \{+1, -1\}$$

$$\mathbf{x} \mapsto y = \delta(\mathbf{x})$$

2.4.1 Learning procedure

Given a set of pre-classified examples (the training set):

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$$

where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$, a learning machine realizes a class of functions $f_{\mathbf{\Lambda}}$, each of that is identified by a set of parameters (the vector $\mathbf{\Lambda}$).

The learning procedure consists in choosing, between those functions, the one that is the most appropriate. Obviously the choice of a function is equivalent to the choice of a set of optimal parameters $\mathbf{\Lambda}$.

Formally, a learning machine is given by the set of functions:

$$f_{\mathbf{\Lambda}} : \quad \mathbb{R} \longrightarrow \{-1, 1\}$$

This procedure has the purpose of choosing the $f_{\mathbf{\Lambda}^*}$ that realizes "the best classification".

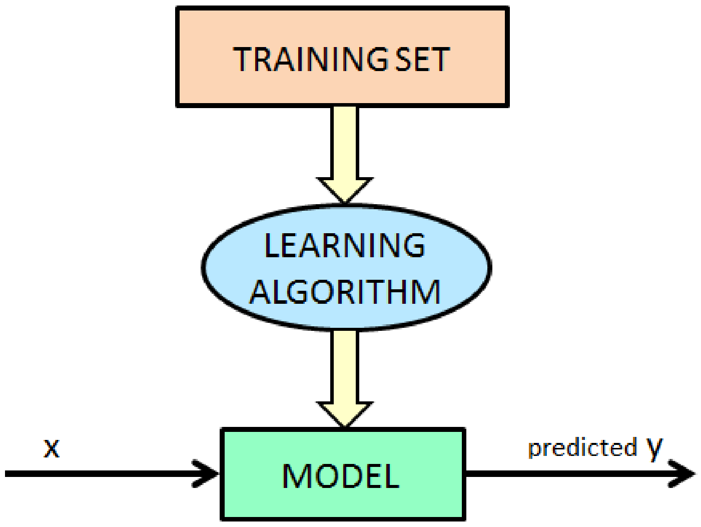


Figure 2.2: Learning procedure scheme.

2.5 Parameters for the evaluation of a classifier

Below we itemize some parameters that enable us to evaluate a particular classifier.

- **ACCURACY**:

  $$Accuracy = \frac{truepositive_{class1} + truepositive_{class-1}}{|class1| + |class-1|}$$

  It represents the percentage of instances correctly classified, whose predicted class coincides with the real one.

- **PRECISION**:

  $$Precision_{class1} = \frac{truepositive_{class1}}{truepositive_{class1} + falsepositive_{class1}}$$

  It is a measure of correctness.

  The lower the number of false positives, the higher (the closer to 1) the precision.

- **RECALL**:

  $$Recall_{class1} = \frac{truepositive_{class1}}{truepositive_{class1} + falsenegative_{class1}}$$

  It is a measure of completeness.

  The lower the number of false negatives, the higher (the closer to 1) the recall.

- **F-MEASURE**:

  $$F - Measure_{class1} = \frac{2 \cdot Recall_{class1} \cdot Precision_{class1}}{Recall_{class1} + Precision_{class1}}$$

2.5.1 *Goodness* and comparison measurements for binary classifiers

Up to now we have talked about "the most appropriated class" an item can be assigned to. It is important to understand on which bases a binary classifier is intended to be a "good" classifier: to choose between the possible $f_{\mathbf{\Lambda}}$ we mentioned in 2.4.1, we have to define a quality criterion.

As we can imagine intuitively, the goodness of a classifier is inversely proportional to the errors it commits (quantitatively and qualitatively).

2.5.1.1 Notation

We will use to talk about classification in an intuitive way: we will consider classification as a data mining problem that aims to determinate the membership of different ***points*** to different ***sets***.

We have a set of points (about some hundreds of points) expressed as vectors of n coordinates in the n-dimensional space.

We have to realize a binary classification for those points. So we have to assign to each point a value in $\{+1, -1\}$, to indicate the membership of the relative item to a specific set, which we call also class.

2.5.1.2 Different possible error measurements

There are several possibilities in measuring that kinds of errors. We itemize the possible error measurements and describe more deeply some of them:

- Empirical error minimization;
- Structural risk minimization;
- Posterior likelihood;
- Percentage of error.

Empirical error minimization.
First of all let us define:

Loss function: $\mathcal{L}(y, f_{\mathbf{\Lambda}}(\mathbf{x})) = \mathcal{L}(y, \mathbf{\Lambda}, \mathbf{x})$
it measures the gap between the predicted value $f_{\mathbf{\Lambda}}(\mathbf{x})$ and the real one y.

Examples of loss function for $f_{\mathbf{\Lambda}}(\mathbf{x}) \, : \, \mathbb{R}^n \longrightarrow \{-1, 1\}$ are:

- Misclassification error, first type:

$$\mathcal{L}(y, f_{\mathbf{\Lambda}}(\mathbf{x})) = \begin{cases} 0 & f_{\mathbf{\Lambda}}(\mathbf{x}) = y \\ 1 & f_{\mathbf{\Lambda}}(\mathbf{x}) \neq y \end{cases}$$

- Misclassification error, second type:

$$\mathcal{L}(y, f_{\mathbf{\Lambda}}(\mathbf{x})) = \frac{1}{2}|f_{\mathbf{\Lambda}}(\mathbf{x}) - y|$$

- Logistic loss:

$$\mathcal{L}(y, f_{\mathbf{\Lambda}}(\mathbf{x})) = ln\left(1 + e^{-(y \cdot f(\mathbf{x}))}\right)$$

What we want to minimize is the *effective risk* or *theoretical error*, committed on the different choices of the vector $\mathbf{\Lambda}$; it can be viewed also as the expected value of the loss we have choosing a particular function.

$$R(\mathbf{\Lambda}) = R(f_{\mathbf{\Lambda}}) = E\left[\mathcal{L}(y, f_{\mathbf{\Lambda}}(\mathbf{x})\right] = \int \mathcal{L}(y, f_{\mathbf{\Lambda}}(\mathbf{x})P(\mathbf{x}, y)d\mathbf{x}dy$$

Since the join probability distribution $P(\mathbf{x}, y)$ is not known, we are not able to calculate the theoretical error. However we know a set of m empirical observations (independent and identically distributed – the data of the training set) that permit us to calculate the empirical error or empirical risk.

$$R_{emp}(f_{\mathbf{\Lambda}}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y_i, f_{\mathbf{\Lambda}}(\mathbf{x}_i))$$

For the ***Big Numbers Law*** we know that

$$\lim_{m \to \infty} R_{emp}(f_{\mathbf{\Lambda}}) = R(f_{\mathbf{\Lambda}})$$

So we can minimize the empirical error instead of the theoretical one.

The function that minimizes the empirical error is not unique. We can decide to choose functions of different complexities, but the complexity degree is related to two particular phenomenons:

- *Overfitting*: when the class of functions $f_{\mathbf{\Lambda}}(\mathbf{x})$ is too complex we will not have a good approximation of the function on the test set;
- *Underfitting*: when the class of functions $f_{\mathbf{\Lambda}}(\mathbf{x})$ is too simple we will not have a good approximation of the function on the training set.

Structural risk minimization.

First of all let us introduce the notion of *VC dimension* (whose name refers to the scientists Vapnik and Chervonenkis) [17].

> **VC dimension** (of a binary classifier): is the maximum number of items of the training that the classifier (with a proper function $f_{\mathbf{\Lambda}}(\mathbf{x})$) is able to separate into two classes; informally it can be intended as a sort of complexity of the classifier.

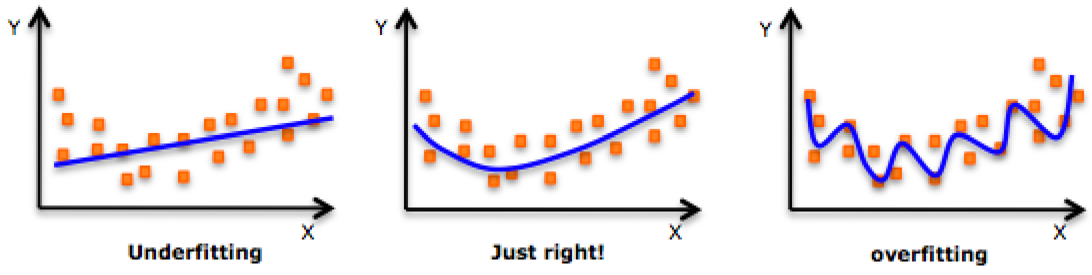


Figure 2.3: Underfitting and overfitting.

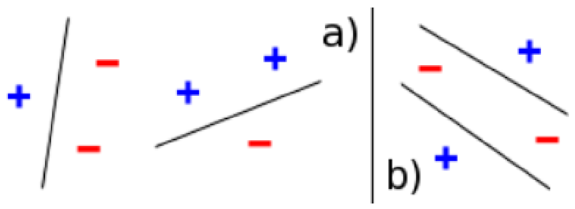


Figure 2.4: **a.** It is always possible to linearly separate three points. **b.** It is not always possible for four points. So the VC dimension in this case is 3.

Below, in figure 2.4, we can see that the VC dimension of a linear classifier in $\mathbb{R}^2$ is 3: in fact we can always separate 3 points with a rect, but no more than 3.

The SVM approach [15] [16] tries to minimize at the same time the empirical error and the complexity of the classifier: this tradeoff approach is called *minimization of the structural risk*, as we can see in figure 2.5.

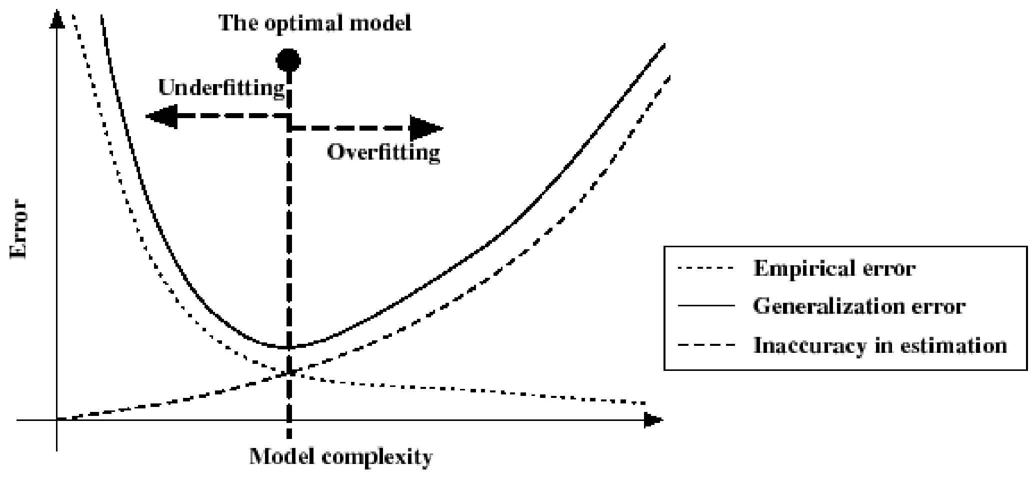


Figure 2.5: Tradeoff of the structural risk minimization.

If m are the points in the training set, h is the VC dimension and $\eta \in (0, 1)$ fixed, an upper bound for that risk, valid with a confidence of $1 - \eta$, is given by

$$R(f_{\mathbf{\Lambda}}) \leq R_{emp}(f_{\mathbf{\Lambda}}) + \Phi(h, m, \eta)$$

where

$$\Phi(h, m, \eta) = \sqrt{\frac{h(log\left(\frac{2m}{h}\right) + 1) - log\left(\frac{\eta}{4}\right)}{m}}$$

Posterior likelihood

This technique measures the probability that the model makes the correct ascription of the items to the relative class, based on the training set data.

$$\mathcal{L}(\mathcal{M}|\mathcal{T}) = \prod_{i=1}^{m} p(y_i|x_i, \mathcal{T}, \mathcal{M})$$

where $\mathcal{T}$ is the training set and $\mathcal{M}$ is the test set.

Typically, for simplicity, is used the logarithm of this measure, we talk about *log-likelihood*

$$\mathcal{L}(\mathcal{M}|\mathcal{T}) = \sum_{i=1}^{m} log[p(y_i|x_i, \mathcal{T}, \mathcal{M})]$$

Percentage of error

Is the simplest and most intuitive measure of error. It is directly linked to the accuracy defined in section 2.5.

Chapter 3

What a SVM is and how it solves the classification problem

3.1 Some history

Support Vector Machines were introduced by Vladimir Vapnik and colleagues (Boser and Guyon) in the late 70s.

The earliest mention was in [Vapnik, 1979], but the first main paper seems to be [Vapnik, 1995].

The theoretical bases were developed from Statistical Learning Theory (Vapnik and Chervonenkis) since the 60s.

Empirically they immediately showed good performances: successful applications in many fields (bioinformatics, text, image recognition, . . .).

3.2 A short description

A Support Vector Machine is defined as

> *A binary classifier that is able to learn the bound between elements that belong to two different classes [19].*

The *SVM approach* can be thought as an alternative learning technique to polynomial, radial basis function and multi-layer perceptron classifiers.

SVMs are sets of supervised learning methods whose training technique permits to represent complex non linear functions. The characteristic parameters of the system are determined solving a quadratic convex optimization problem. That SVM has the "mathematical advantage" of having a global minimum: it ensures that the resulting parameters are actually the best that can be found for the problem, given the particular training set.

The purpose of SVM is to perform a classification by constructing a n-dimensional hyperplane that optimally separates the data (the points) into two categories.

With reference to the description we will do in 3.3.2.2 about non linear classifiers on non linearly separable data, SVM were originally defined for the classification in classes of objects that are linearly separable, but obviously they can be used also to separate classes of elements non linearly separable, making them really interesting in the scientific environment.

Summarizing, the functionality of a SVM depends on three factors:

The kernel type specific for a particular problem: it enables the system to classify properly also non linearly separable data;

The optimization model that works on the training set, but provides robust values $\mathbf{\Lambda}$ and b parameters, that can be adequate also to the classification of the test set items;

The setting of parameters γ and C that is realized by a trial-and-error validation procedure and represents the most difficult part in the use of SVM.

For a mathematical description of *SVM approach* see section 3.3 [7] [8] [9] [10] [11].

3.3 How a SVM solves the classification problem

Given the training set

$$T = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^n,\ y_i \in \{-1, +1\},\ i = 1, \dots, m\}$$

and a properly tuned parameter C, the *SVM approach* solves the following optimization problem

$$\begin{aligned} min_{\xi,b,\mathbf{w}}\ & C\sum_{i=1}^{m} \xi_i + \frac{1}{2}||\mathbf{w}||^2 \\ & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \dots, m \\ & \xi_i \geq 0 \quad i = 1, \dots, m \\ & b \in \mathbb{R} \\ & \mathbf{w} \in \mathbb{R}^n \end{aligned}$$

We will deal with the dual of the problem above, which is

$$\begin{aligned} max_\lambda\ & \sum_{i=1}^{l} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{l} \lambda_i\lambda_j y_i y_j \mathbf{x}_i\mathbf{x}_j \\ & \sum_{i=1}^{l} \lambda_i y_i = 0 \\ & 0 \leq \lambda_i \leq C \quad i = 1, \dots, l \end{aligned}$$

To understand the theory the model above is based on, we have to clarify how to deal with linearly and non linearly separable data.

3.3.1 Classifier on linearly separable data

We want to find the hyperplane H, identified by the parameters $(\mathbf{w}, b)$, that separates the data in the best way: it has to be as far as possible by each point $\mathbf{x}_i$ to be separated.

For the training set represented in figures below, figure 3.1 shows one of the infinite hyperplanes that separate the linearly separable points of the dataset, and figure 3.2 shows the optimum one among them.

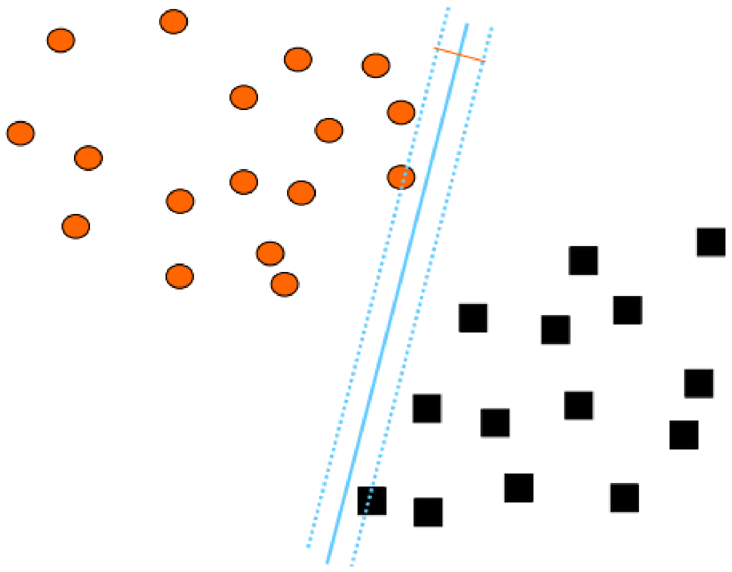


Figure 3.1: A generical separating hyperplane.

The distance between a point $\mathbf{x}$ and hyperplane $(\mathbf{w}, b)$ is:

$$d(\mathbf{x}, (\mathbf{w}, b)) = \frac{|\mathbf{w}^T \cdot \mathbf{x} + b|}{||\mathbf{w}||}$$

Explanation of the result.
Let us explain the result got above, helping with figure 3.3.

We have:

- A hyperplane H, defined by a linear discriminant function:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$

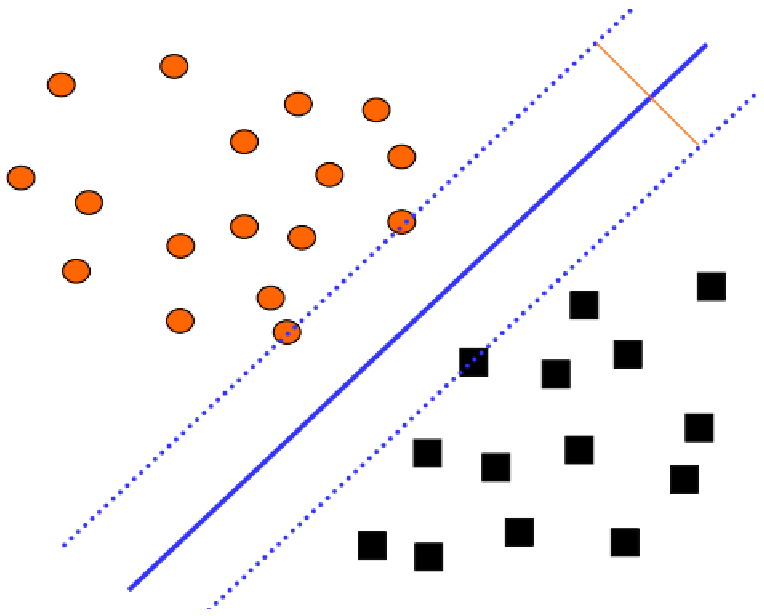


Figure 3.2: The optimum separating hyperplane.

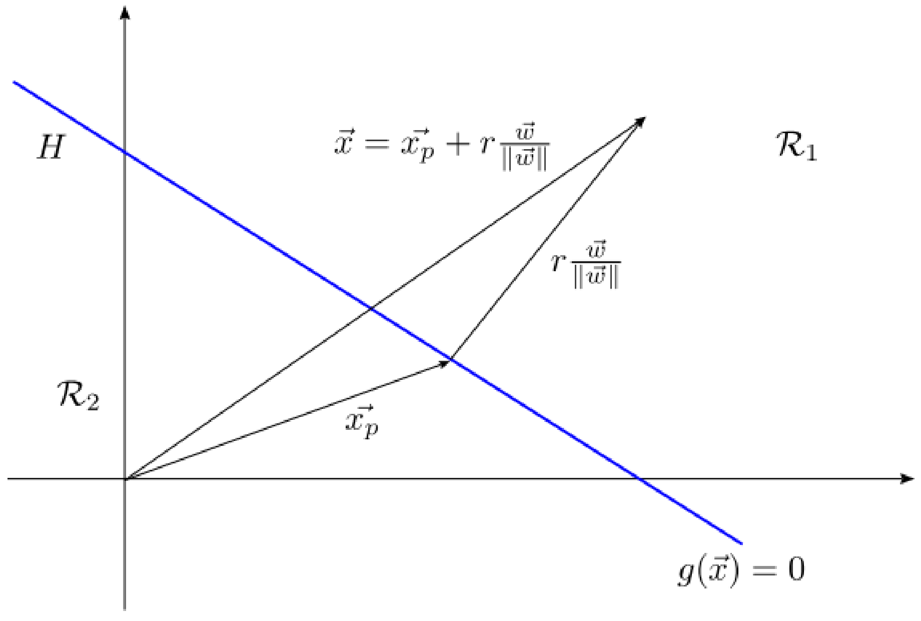

H
$\vec{x} = \vec{x_p} + r\frac{\vec{w}}{\|\vec{w}\|}$
$\mathcal{R}_1$
$r\frac{\vec{w}}{\|\vec{w}\|}$
$\mathcal{R}_2$
$\vec{x_p}$
$g(\vec{x}) = 0$

Figure 3.3: Distance point $\mathbf{x}$ - hyperplane $g(\mathbf{x}) = 0$.

- A point $\mathbf{x}$, defined by a n-dimensional vector, whose distance from H we want to calculate.

Using the vectorial sum we can express $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{x}_p + \mathbf{x}_n$$

where $\mathbf{x}_p$ is the component parallel to the hyperplane and $\mathbf{x}_n$ is the normal one;

- A unitary norm vector, normal to the hyperplane H

$$\mathbf{w}' = \frac{\mathbf{w}}{||\mathbf{w}||}$$

.

The parallel component $\mathbf{x}_p$ is the orthogonal projection of $\mathbf{x}$ on the hyperplane H.

We can express the component that is normal to the hyperplane as

$$\mathbf{x}_n = r \cdot \mathbf{w}' = r \cdot \frac{\mathbf{w}}{||\mathbf{w}||}$$

Where r is the algebraic distance between $\mathbf{x}$ and H.

So we can write

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

Using that decomposition of $\mathbf{x}$ we find that

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

$$= \mathbf{w}^T\mathbf{x}_p + b + r \cdot \frac{\mathbf{w}^T\mathbf{w}}{||\mathbf{w}||} =$$

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

Since the vector $g(\mathbf{x}_p)$ lies on the hyperplane (and so we have that $g(\mathbf{x}_p) = 0$) and since the inner product $\mathbf{w}^T\mathbf{w}$ is equal to $||\mathbf{w}||^2$, we can simplify the expression above finding that

$$g(\mathbf{x}) = r \cdot ||\mathbf{w}||$$

And so that

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

Since a distance is always expressed as a positive value

$$r = \frac{|\mathbf{w}^T\mathbf{x} + b|}{||\mathbf{w}||}$$

That is what we wanted to explain.

The optimal hyperplane is the one that has the maximum distance to the closest points. So we have to deal with a max-min optimization function

$$max_{\mathbf{w},b}\, min_{1\leq i\leq n} d(\mathbf{x}_i, (\mathbf{w}, b))$$

We have a parametrization that is function of $||\mathbf{w}||$: we have to fix the value of $||\mathbf{w}||$. There are multiple choices:

- $||\mathbf{w}|| = 1$
- $||\mathbf{w}||$ such that $min_{1\leq i\leq n} d(\mathbf{x}_i, (\mathbf{w}, b)) = \frac{1}{||\mathbf{w}||}$

In the second case the condition imposed is equivalent to the following

$$min_{1\leq i\leq n}|\mathbf{w}^T\mathbf{x}_i + b| = 1$$

that comes from

$$min_{1\leq i\leq n}\frac{|\mathbf{w}^T\mathbf{x} + b|}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||}$$

The algebraic manipulation we made, permits to reduce the problem to a quadratic programming problem.

Let us consider the points $\mathbf{x}_{\pi_k},\ k = 1, \ldots, t$ where $\{\pi_k,\ \ k = 1, \ldots, t\} \subseteq \{i = 1, \ldots, m\}$ of the system that have the minimum distance to the hyperplane H. We call that points ***support vectors***. As we said before, for each support vector $\mathbf{x}_i$ we have

$$\mathbf{x}_i\ :\ |\mathbf{w}^T\mathbf{x}_i + b| = 1$$

We introduce the concept of ***margin*** M, that refers to the distance between the hyperplanes defined by the support vectors (they are the hyperplanes that are parallel to the hyperplane H and that have in common with the specific support vector the point defines by its coordinates).

$$M = \left|\frac{1}{||\mathbf{w}||} - \frac{-1}{||\mathbf{w}||}\right| = \frac{2}{||\mathbf{w}||}$$

Our purpose is to maximize the margin, satisfying the conditions that define $\mathbf{w}$ and b.

Since maximizing the margin is equivalent to minimizing the reciprocal of the margin, so we can formulate the problem as defined below

$$\begin{aligned} &min\ \frac{1}{2}||\mathbf{w}^T\mathbf{w}|| \\ &y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad i = 1, 2, \ldots, m \\ &b \in \mathbb{R} \\ &\mathbf{w} \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

The Lagrangian relaxation of the given problem is

$$\begin{aligned} min \ & \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \lambda_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \\ & b \in \mathbb{R} \\ & \mathbf{w} \in \mathbb{R}^n \end{aligned}$$

The solution of this relaxation is obtained applying the Karush-Kuhn-Tucker (KKT) conditions to the problem in (3.1).

We recall them in the following [12]

Theorem [KKT conditions]:

Suppose that the objective function $f : \mathbb{R}^n \to R$ and the constraint function $g_i : \mathbb{R}^n \to R$ are continuously differentiable at a point $\mathbf{w}^*$.
Assume $\mathbf{w}^*$ is a regular local minimum of a nonlinear programming problem.
Then there is a Lagrange multiplier vector $\mathbf{\Lambda}$ such that

$$\nabla f(\mathbf{w}^*) = \sum_{i=1}^{m} {\lambda_i}^* \nabla g_i(\mathbf{w}^*)$$

$${\lambda_i}^* \geq 0 \quad i = 1, \ldots, m$$

$$g_i(\mathbf{w}^*) \geq 0 \quad i = 1, \ldots, m$$

$${\lambda_i}^* g_i(\mathbf{w}^*) = 0 \quad i = 1, \ldots, m$$

Note that

$$\nabla f(\mathbf{w}^*) = \sum_{i=1}^{m} {\lambda_i}^* \nabla g_i(\mathbf{w}^*) \iff \nabla \mathcal{L}(\mathbf{w}, b, \mathbf{\Lambda}) = 0$$

Necessary and sufficient conditions for

$$\nabla \mathcal{L}(\mathbf{w}, b, \mathbf{\Lambda}) = 0$$

are

$$\mathbf{w}^* = \sum_{i=1}^{m} y_i {\lambda_i}^* \mathbf{x}_i$$

$$\sum_{i=1}^{m} y_i {\lambda_i}^* = 0$$

The complete formulation of the Lagrangian relaxation is

$$\begin{aligned}
&min\ \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \lambda_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1)\\
&\sum_{i=1}^{m} \lambda_i y_i = 0\\
&\lambda_i \geq 0 \quad i = 1, \ldots, m\\
&b \in \mathbb{R}\\
&\mathbf{w} \in \mathbb{R}^n
\end{aligned}$$

The objective function can be properly manipulated, using the necessary and sufficient conditions previously obtained. That is, knowing that the hyperplanes can be written as linear combinations of the vectors of the training set ($\mathbf{w} = \sum_{i=1}^{m} \lambda_i y_i \mathbf{x}_i$) and using the second condition that implies that $\sum_{i=1}^{m} \lambda_i y_i b = 0$.

$$min \left[\frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \lambda_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1)\right] =$$

$$= min \left[\frac{1}{2}\sum_{i=1}^{m} \lambda_i \mathbf{x}_i y_i \sum_{j=1}^{m} \lambda_j \mathbf{x}_j y_j - \sum_{i=1}^{m} \lambda_i \mathbf{x}_i y_i \sum_{j=1}^{m} \lambda_j \mathbf{x}_j y_j - \sum_{i=1}^{m} \lambda_i y_i b + \sum_{i=1}^{m} \lambda_i\right] =$$

$$= min \left[-\frac{1}{2}\sum_{i=1}^{m} \lambda_i \mathbf{x}_i y_i \sum_{j=1}^{m} \lambda_j \mathbf{x}_j y_j + \sum_{i=1}^{m} \lambda_i\right]$$

The Lagrangian relaxation has to be minimized on $\mathbf{w}$ and b and to be maximized on $\mathbf{\Lambda}$.

So we can formulate the problem as in the following

$$\begin{aligned}
&max_{\mathbf{\Lambda}}\ \sum_{i=1}^{m} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \mathbf{x}_j\\
&\sum_{i=1}^{m} \lambda_i y_i = 0\\
&\lambda_i \geq 0 \quad i = 1, \ldots, m
\end{aligned}$$

In the model above the bounds on $\mathbf{w}$ and b are replaced by bounds on the Lagrangian multipliers and the training set vectors appear only as inner products between vectors.

Since the equation of the optimal hyperplane can be written as a linear combination of the vectors of the training set

$$\mathbf{w}^* = \sum_i \lambda_i^* y_i \mathbf{x}_i = \mathbf{\Lambda}^* y\, \mathbf{x}$$

Then

$$\mathbf{w}^*\mathbf{x} + b = \mathbf{\Lambda}^* y\, \mathbf{x} \cdot \mathbf{x} + b$$

So the classifier is given by

$$sign\left[\sum_{i=1}^{m} y_i \lambda_i^* (\mathbf{x} \cdot \mathbf{x_i}) + b^*\right]$$

where we can find the value of b^* from the following conditions

$$\lambda_i^* \left(y_i(\mathbf{w}^* \mathbf{x} + b^*) - 1\right) = 0 \quad i = 1, \ldots, m$$

$$\longrightarrow \quad b^* = y_i - \mathbf{w}^* \cdot \mathbf{x}_i$$

Eventually, it is possible to demonstrate that [13]

$$b^* = -\frac{1}{2}\left[max_{y_i=-1}\mathbf{w}^{*T}\mathbf{x}_i + min_{y_i=+1}\mathbf{w}^{*T}\mathbf{x}_i\right]$$

In the solution, the points that have the correspondent Lagrangian multipliers $\lambda_i > 0$ are the support vectors; the other points of the training set have the correspondent $\lambda_i = 0$ and so they do not influence the classifier (we could consider just the support vectors' points – the whole information about the training set is contained in those points).

3.3.2 Classifier on non linearly separable data

When datasets are not linearly separable (see figure 3.5), we can either decide to use linear classifier, or to use non linear classifier.

We should prefer the use of the second type of classifiers in order to perform a more accurate classification. Obviously its use makes stuffs also more complicated, as we will see in the following.

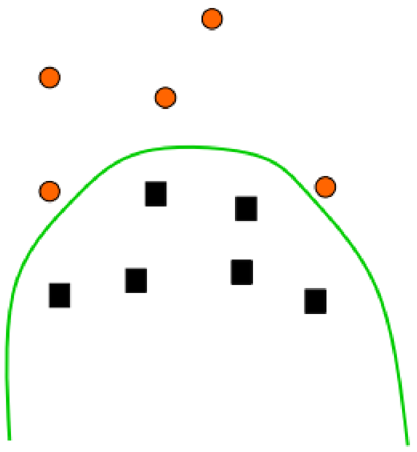


Figure 3.4: Non linearly separable dataset.

3.3.2.1 Linear classifier on non linearly separable data

The linear classifier described above cannot deal with dataset that are non linearly separable.

However it is possible to use a linear classifier (see figure 3.5) also on this kind of data: we just have to relax the classification bounds, tolerating a certain number of errors.

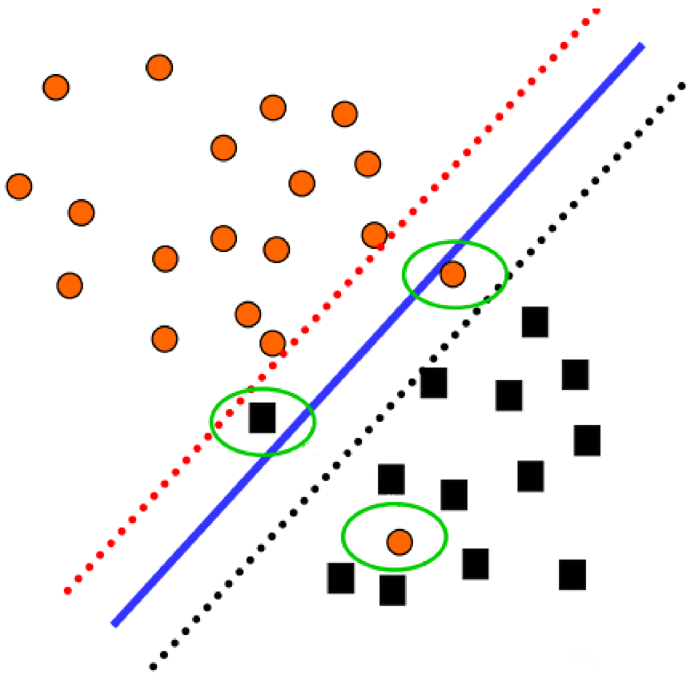


Figure 3.5: Non linearly separable dataset with a linear separation.

The optimum hyperplane is again determined by the support vectors points, but there are some points that do not satisfy the condition $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$. A possible solution is to add some *slack variables.*

The new bounds are

$$\begin{aligned}
&y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, m \\
&\xi_i \geq 0 \quad i = 1, \ldots, m \\
&b \in \mathbb{R} \\
&\mathbf{w} \in \mathbb{R}^n
\end{aligned}$$

where $\xi_i > 1$ represents the error that occurs.

We can reformulate the problem with a model that tries at the same time to minimize $||\mathbf{w}||$ and minimum number of errors (in the objective function there is a term $\sum_{i=1}^{m} \xi_i$ that represents an upper bound on the number of errors on the training data).

$$\begin{aligned}
& min\ \frac{1}{2}||\mathbf{w}||^2 + C(\sum_{i=1}^{m} \xi_i)^k \\
& y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \dots, m \\
& \xi_i \geq 0 \quad i = 1, \dots, m \\
& b \in \mathbb{R} \\
& \mathbf{w} \in \mathbb{R}^n
\end{aligned}$$

where C and k are parameters that are related to the penalty that occurs in case of an error, and typically is $k = 1$.

Recalling that $\mathbf{w} = \sum_{i=1}^{m} \lambda_i y_i \mathbf{x}_i$, an equivalent formulation of the primal problem (for $k = 1$), which we used in our experimentation, is

$$\begin{aligned}
& min\ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i\lambda_j y_i y_j \mathbf{x}_i\mathbf{x}_j + C\sum_{i=1}^{m} \xi_i \\
& y_i\left(\sum_{j=1}^{m} \lambda_j y_j \mathbf{x}_j \cdot \mathbf{x}_i + b\right) \geq 1 - \xi_i \quad i = 1, \dots, m \\
& \xi_i \geq 0 \quad i = 1, \dots, m \\
& \lambda_i \geq 0 \quad i = 1, \dots, m \\
& b \in \mathbb{R}
\end{aligned}$$

As in 3.3.1, the Lagrangian relaxation of the given problem is

$$min\left[\frac{1}{2}||\mathbf{w}||^2 + C(\sum_{i=1}^{m} \xi_i)^k - \sum_{i=1}^{m} \lambda_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{m} \gamma_i\xi_i\right]$$

Under the following constraint, obtained applying *KKT conditions* to the problem formulation

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

and

$$\begin{aligned}
& \xi_i \geq 0 \quad i = 1, \dots, m \\
& \lambda_i \geq 0 \quad i = 1, \dots, m \\
& b \in \mathbb{R} \\
& \mathbf{w} \in \mathbb{R}^n
\end{aligned}$$

The objective function can be properly manipulated, using the necessary and sufficient conditions previously obtained. That is, knowing that the hyperplanes can be written as linear combinations of the vectors of the training set ($\mathbf{w} = \sum_{i=1}^{m} \lambda_i y_i \mathbf{x}_i$) and using the condition $\sum_{i=1}^{m} \lambda_i y_i = 0$ that implies that $\sum_{i=1}^{m} \lambda_i y_i b = 0$.

$$min \left[\frac{1}{2}||\mathbf{w}||^2 + C(\sum_{i=1}^{l} \xi_i)^k - \sum_{i=1}^{m} \lambda_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{m} \gamma_i \xi_i \right] =$$

$$= min \left[\frac{1}{2} \sum_{i=1}^{m} \lambda_i \mathbf{x}_i y_i \sum_{j=1}^{m} \lambda_j \mathbf{x}_j y_j + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \lambda_i \mathbf{x}_i y_i \sum_{j=1}^{m} \lambda_j \mathbf{x}_j y_j + \sum_{i=1}^{m} \lambda_i - \sum_{i=1}^{m} \lambda_i \xi_i + \sum_{i=1}^{m} \gamma_i \xi_i \right] =$$

$$= min \left[-\frac{1}{2} \sum_{i=1}^{m} \lambda_i \mathbf{x}_i y_i \sum_{j=1}^{m} \lambda_j \mathbf{x}_j y_j + \sum_{i=1}^{m} \lambda_i + \sum_{i=1}^{m} (C - \lambda_i - \gamma_i) \xi_i \right]$$

It is possible to demonstrate that $\sum_{i=1}^{m} (C - \lambda_i - \gamma_i) \xi_i = 0$.

The Lagrangian relaxation has to be minimized on $\mathbf{w}$ and b and to be maximized on λ.

So we can formulate the problem as in the following

$$\begin{aligned} & max_\lambda \; \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \mathbf{x}_j \\ & \sum_{i=1}^{m} \lambda_i y_i = 0 \\ & 0 \le \lambda_i \le C \quad i = 1, \dots, m \end{aligned}$$

And, as in 3.3.1, the classifier is given by

$$sign \left[\sum_{i=1}^{m} y_i \lambda_i^* (\mathbf{x} \cdot \mathbf{x}_i) + b^* \right]$$

3.3.2.2 Non linear classifier on non linearly separable data

Another possibility to deal with non linearly separable data is to use a non linear classifier.

The idea behind this approach is to create a sort of "lifting" (using a function Φ) of the non linearly separable data on a new space (generally having more dimensions than the previous one) where they are become linearly separable.

$$\Phi \, : \, \mathbb{R}^n \mapsto \mathbb{R}^N \quad N \gg n$$

$$\mathbf{x} \mapsto \Phi(\mathbf{x})$$

Through this procedure (known in the literature as *kernel trick* [14]) we can use once again a linear classifier in the new space, as seen before.

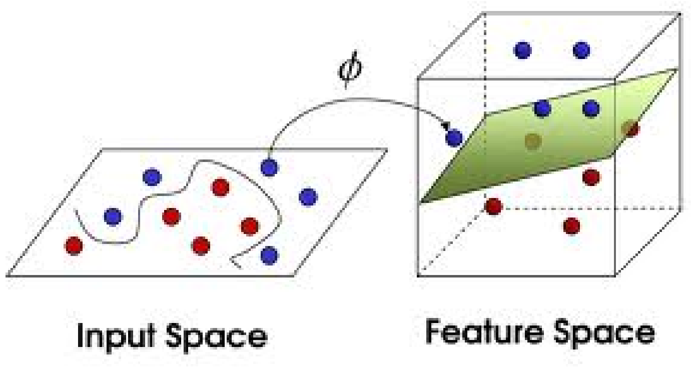


Figure 3.6: The SVM trick for non linearly separable data.

The previous equation for the classifier

$$sign\left[\sum_{i=1}^{m} y_i\lambda_i^*(\mathbf{x}\cdot\mathbf{x}_i)+b^*\right]$$

is changed in the following equation

$$sign\left[\sum_{i=1}^{m} y_i\lambda_i^*(\Phi(\mathbf{x})\cdot\Phi(\mathbf{x}_i))+b^*\right]$$

That model has a limit: the product $\Phi(\mathbf{x})\cdot\Phi(\mathbf{x}_i)$ involves high dimensional vectors. A solution here is to introduce a kernel function of the form

$$K\ :\ \mathbb{R}^n\times\mathbb{R}^n\ \mapsto\ \mathbb{R}$$

that substitutes the product $K(\mathbf{x}_i,\mathbf{x}_j)=\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$.

The characteristics and properties of the kernel function are defined by

Mercer Theorem:

A symmetric function $K(\mathbf{x}_i,\mathbf{x}_j)$ is a kernel
if and only if
for any sample $S=\{\mathbf{x}_1,\ldots,\mathbf{x}_m\}$, the kernel matrix for S is positive semi-definite.

Using the definition of the kernel function in the classifier we have

$$sign\left[\sum_{i=1}^{m} y_i\lambda_i^*K(\mathbf{x},\mathbf{x}_i)+b^*\right]$$

And in the model

$$
\begin{aligned}
max\ & \sum_{i=1}^{m} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{m} \lambda_i\lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\
& \sum_{i=1}^{m} \lambda_i y_i = 0 \\
& 0 \leq \lambda_i \leq C \quad i = 1, \ldots, m
\end{aligned}
$$

Or equivalently for the primal problem

$$
\begin{aligned}
min\ & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i\lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) + C\sum_{i=1}^{m} \xi_i \\
& y_i\left(\sum_{j=1}^{m} \lambda_j y_j K(\mathbf{x}_i, \mathbf{x}_j) + b\right) \geq 1 - \xi_i \quad i = 1, \ldots, m \\
& \xi_i \geq 0 \quad i = 1, \ldots, m \\
& \lambda_i \geq 0 \quad i = 1, \ldots, m \\
& b \in \mathbb{R}
\end{aligned}
$$

How to choose the kernel function?

Kernel methods work:

- mapping items on a different vectorial space;
- looking for linear relationships between items in that space.

Here there are several possibilities for the kernel method:

Linear:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{m} (\mathbf{x}_i)_k \cdot (\mathbf{x}_j)_k$$

Polynomial:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \sum_{k=1}^{m} (\mathbf{x}_i)_k \cdot (\mathbf{x}_j)_k\right)^d$$

Radial Based function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\sum_{k=1}^{m}((\mathbf{x}_i)_k - (\mathbf{x}_j)_k)^2}{2\sigma^2}}$$

Gaussian:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma\left[\sum_{k=1}^{m}((\mathbf{x}_i)_k - (\mathbf{x}_j)_k)^2\right]} = e^{-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2}$$

Multi-Layer Perceptron:

$$K(\mathbf{x}_i, \mathbf{x}_j) = tanh \left[b \left(\sum_{k=1}^{m} (\mathbf{x}_i)_k (\mathbf{x}_j)_k \right) - c \right]$$

However, there are not limits to the choice of the kernel function; necessary and sufficient condition is that, since that functions have to represent an inner product in the extended space, kernel functions satisfy the following properties:

1. $K(\mathbf{x}_1, \mathbf{x}_2) = K(\mathbf{x}_2, \mathbf{x}_1)$
2. $K(\mathbf{x}_1, \mathbf{x}_2 + \mathbf{x}_3) = K(\mathbf{x}_1, \mathbf{x}_2) + K(\mathbf{x}_1, \mathbf{x}_3)$
3. $K(\mathbf{x}_1, \alpha\mathbf{x}_2) = \alpha K(\mathbf{x}_1, \mathbf{x}_2)$

The most used kernel is the Gaussian one. It is different from the other types of kernel because of its meaning: it represents a function whose value, given two points in input, depends only on the distance between them, without any dependency on the absolute position of the points in the whole set of points.

To understand the meaning and behaviour of Gaussian kernel, let us consider two points in the original space: x_i and x_j.

- If the two points coincide, then $d(\mathbf{x}_i, \mathbf{x}_j) = 0$, then $K(\mathbf{x}_i, \mathbf{x}_j) = 1$;
- If not, the trend of K is strictly decreasing in the amount of distance (the function trend is given in figure 3.7, where the x-axis represents the distance and the y-axis the value of K, for Gaussian kernel).

An analogy to figure with Gaussian kernel.

To go deeply inside an intuitive comprehension of the behaviour of Gaussian kernel, a useful analogy can be given.

Each point $\mathbf{x}$ of the training set is like a communication source that transmits a particular signal $f(\mathbf{x})$ uniquely associated to it, namely the constant signal $+1$ or -1. This signal is manipulated externally (this is the contribution of the Gaussian kernel) so that, after its emission, it is decreased with an exponential law, depending on the distance (far from the transmitting point, the signal is close to 0, near the point the signal is close to 1).

When we want to classify a new point $\mathbf{x}^1$, not belonging to the training set, we perform the following actions: standing in $\mathbf{x}^1$, we measure the whole perceived signals, computing the total signed sum: if the result is positive, then $\mathbf{x}^1$ is classify as a point that belongs to class $+1$, otherwise if the result is negative $\mathbf{x}^1$ belongs to -1.

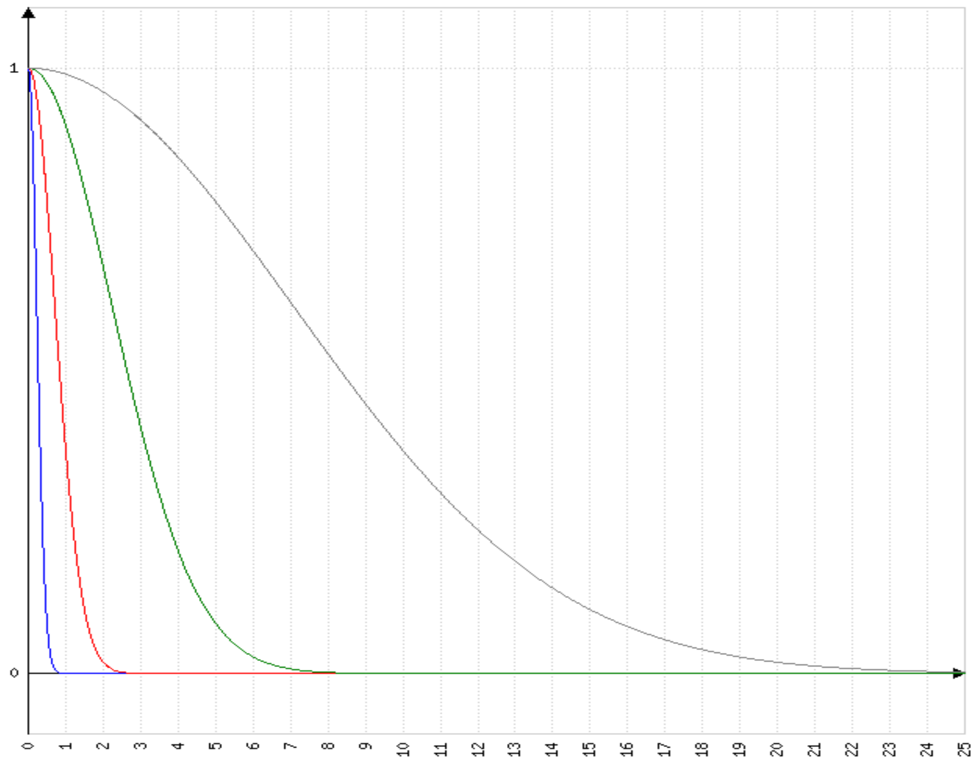


Figure 3.7: Gaussian kernel value *vs* distance, for the parameter $\gamma = 0.1, 1, 10, 100$.

Clearly, the density of the training set points is strictly related to the determination of the constant (the value of γ in the exponential Gaussian function) that defines the decay of the signal with the distance: if points are close together, then signals can decrease in a less intense way than in the case they are far each from the others (in this last case, if signals decrease too quickly there is the risk that the points do not get any signal from the others).

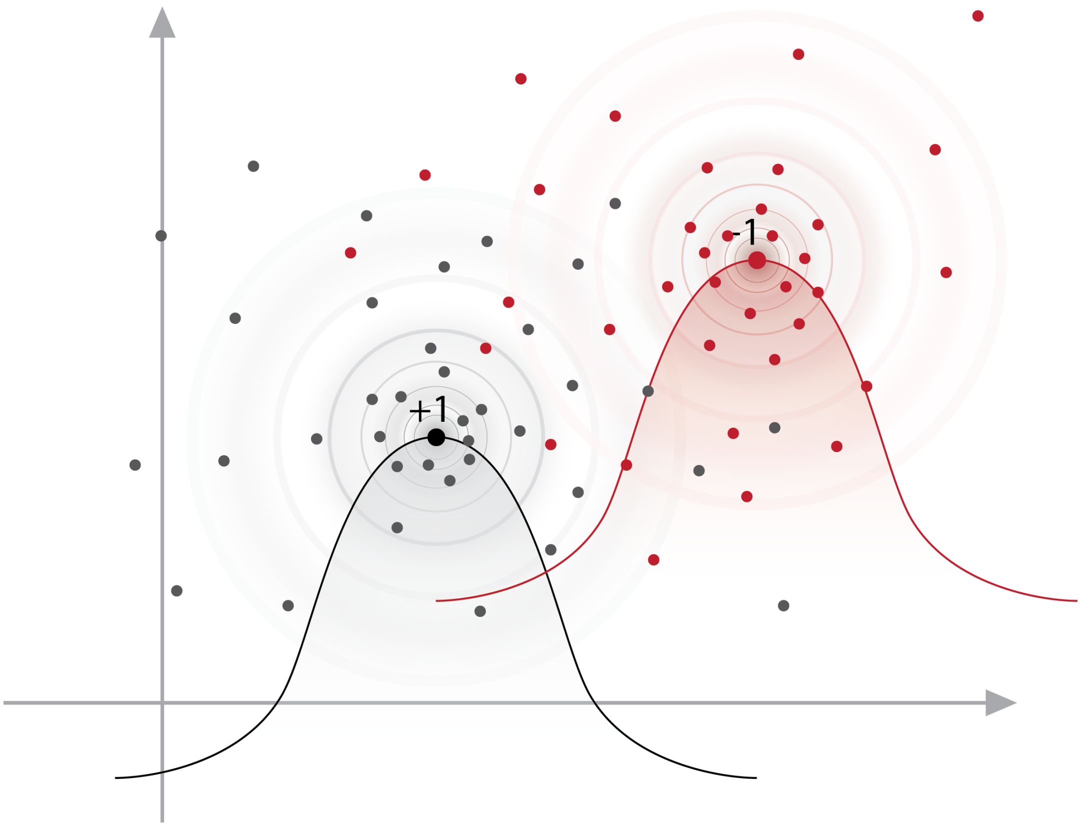


Figure 3.8: Signals analogy.

Chapter 4

Alternative training models

4.1 Gaussian kernel models

In this section we will describe the mathematical models we implemented as alternative approaches to the classical quadratic programming model used by the SVM approach, namely:

$$
\begin{aligned}
&min\ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)+C\sum_{i=1}^{m}\xi_i \\
&\quad y_i\left(\sum_{j=1}^{m}\lambda_j y_j K(\mathbf{x}_i,\mathbf{x}_j)+b\right)\geq 1-\xi_i \quad i=1,\dots,m \\
&\quad \xi_i\geq 0 \quad i=1,\dots,m \\
&\quad \lambda_i\geq 0 \quad i=1,\dots,m \\
&\quad b\in\mathbb{R}
\end{aligned}
\tag{4.1}
$$

where $K(\mathbf{x}_j,\mathbf{x}_i)$ is the considered kernel, e.g. the Gaussian one

$$K(\mathbf{x}_i,\mathbf{x}_j)=e^{-\gamma||\mathbf{x}_i-\mathbf{x}_j||^2}$$

We recall that the resolution of the proper mathematical model enables a SVM to determinate the optimum $\mathbf{\Lambda}$ and b parameters to be used in phase of classification. The aforesaid model is structured on the training set points.

About the models that will be proposed in the following, we also recall that

*A **MIP problem** (Mixed Integer Programming) consists in the minimization of a linear objective function with a finite number of linear constraints, with an additional constraint that requires some/all variables to be integer [26].*

This kind of models have to be solved with proper `CPLEX` settings. In the `.dat` file, where we define the parameters setting to be read by `CPLEX` before the optimization, we have the following "rules":

```
set mip tolerance integer 0
set mip timelimit 1200
set mip polishaftertime 900
```

where:

- the first line refers to the tolerance considered by the solver on the constrains;
- the second line sets a timelimit of 20 minutes for the optimization process;
- the third line establishes to start a proper procedure, called *polish*, after the passage of $\frac{3}{4}$ of the timelimit interval.

4.1.1 SVM_aINT

In this model we impose λ_i variables to be integer and also we impose an upper bound for their values.

$$\begin{aligned}
&min\ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\lambda_i\lambda_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) + C\sum_{i=1}^{m}\xi_i\\
&y_i\left(\sum_{j=1}^{m}\lambda_j y_j K(\mathbf{x}_i,\mathbf{x}_j)+b\right) \geq 1-\xi_i \quad i=1,\dots,m\\
&\xi_i \geq 0 \quad i=1,\dots,m\\
&0 \leq \lambda_i \leq R \quad integer \quad i=1,\dots,m\\
&b \in \mathbb{R}
\end{aligned} \tag{4.2}$$

where $R \in \{1, 10, 100\}$.

This model is aimed to limit overfitting on λ_i values. As a matter of facts, this phenomenon tends to produce high accuracies on the validation set, but actually gives worse final accuracies over the test set.

The model realizes its purpose imposing λ_i values to belong to a finite subset of possible values (through the simultaneous imposition of integer values and upper bound for λ_i).

4.1.2 SVM_TV

In this model we train the system not on the overall training set, but on just a $\frac{4}{5}$ portion of it. At the same time we try to minimize, though the minimization of the objective function, the number of errors on the complementary $\frac{1}{5}$ portion of the training set.

$$
\begin{aligned}
min \ & \sum_{j=1}^{l} z_j \\
& y_j \left(\sum_{i=1}^{n} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b \right) \geq 1 - M \cdot z_j \quad j = 1, \ldots, l \\
& z_j \in \{0, 1\} \quad j = 1, \ldots, l \\
& \lambda_i \geq 0 \quad i = 1, \ldots, n \\
& b \in \mathbb{R}
\end{aligned}
\tag{4.3}
$$

where $V \subset \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ is the validation set, and $T = TrainingSet - V$.

As we can see, we train the system over V, obtaining the relative $\mathbf{\Lambda}$ and b values, and, at the same time, we test our approach on T, through the count the committed errors.

This way we will obtain at the end just $\frac{4}{5}$ of the λ_i variables, but we hope the mixed procedures helps us to reduce the overfitting of the variables on the training set items.

This problem has a particularity: it does not use the parameter C, that appears in the objective function of the classic SVM quadratic programming model. Therefore, its validation procedure is not aimed to the determination of the optimum parameters pair (C, γ), but actually to the determination of the combination between parameter γ and training set division that gives the best training accuracy.

4.1.3 SVM_TV_aINT

In the current section we propose a model that is a variant of the one described in section 4.1.2, but here we impose also variables λ_i to be integer, trying again to reduce overfitting.

$$
\begin{aligned}
min\ & \sum_{j=1}^{m} z_j \\
& y_j \left(\sum_{i=1}^{n} \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b \right) \geq 1 - M \cdot z_j \quad j = 1, \ldots, m \\
& z_j \in \{0, 1\} \quad j = 1, \ldots, m \\
& 0 \leq \lambda_i \leq R \ \ integer \quad i = 1, \ldots, m \\
& b \in \mathbb{R}
\end{aligned} \tag{4.4}
$$

The same consideration made previously for the characteristics of SVM_TV model are valid also for this model. In addiction, the restricted possible values imposed for variables λ_i represent a further attempt to the reduction of overfitting.

4.1.4 SVM_aAll_TV

This is the same model as in section 4.1.2, but, in this case, we use all the variables λ_k in the final classifier, integrating the missing λ_j (those that were not determined by the resolution of the model) with the mean value calculated on the given λ_i.

With this model we want to improve the performances of SVM_TV using more variables than those used by the aforesaid model.

4.1.5 SVM_Mixed

In this model we accept to incur in a number of errors during the classification of the training items that is at most equal to a certain percentage of the total number of training items themselves.

$$
\begin{aligned}
min\ & \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\
& y_i \left(\sum_{j=1}^{m} \lambda_j y_j K(\mathbf{x}_j, \mathbf{x}_i) \right) \geq 1 - M \cdot z_i \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} z_i \leq C \cdot m \\
& z_i \in \{0, 1\} \quad i = 1, \ldots, m \\
& \lambda_i \geq 0 \quad i = 1, \ldots, m
\end{aligned} \tag{4.5}
$$

where $C \in \{1, 10\}$.

With this model we want again to reduce the overfitting phenomenon on the training set. We aim to do that through the action of the constraint that enables the system to commit a number of errors that is even higher than the number committed by the classic SVM approach model. This action reduces the accuracies on the validation set, that is it makes the variables less fitting to the validation set itself.

Besides, another interesting observation is that we try to minimize the total misclassification not through the minimization of the sum of ξ_i variables, but yet through the minimization of the number of misclassified points (represented by z_i variables).

4.1.6 SVM_MixedMu

This model is a variant of the one proposed in section 4.1.5.

$$\begin{aligned}
min \ & \sum_{i=1}^{m} z_i \\
& y_i \left(\sum_{j=1}^{m} \lambda_j y_j K(\mathbf{x}_j, \mathbf{x}_i) \right) \geq C \cdot \mu - M \cdot z_i \quad i = 1, \dots, m \\
& \sum_{i=1}^{m} \lambda_i \leq \mu \cdot m \\
& z_i \in \{0, 1\} \quad i = 1, \dots, m \\
& \lambda_i \geq 0 \quad i = 1, \dots, m \\
& \mu \geq 1
\end{aligned} \tag{4.6}$$

where $C \in \{1, 10\}$.

In this model we substitute the quadratic objective function of the one in section 4.1.5 with a linear one that tries to minimize the number of errors we incur in on the test set items.

Besides, we substitute the 1, that appears in the right side of the first set of constraints, with the mean of the λ_i values.

4.2 Linear kernel models

In this section we will describe models that are intended to use the linear kernel. Hence, they are expressed in terms of $(\mathbf{w}, b)$, instead of $\mathbf{\Lambda}$ and b.

4.2.1 SVM_K-lin

This is the classical quadratic programming problem, using linear kernel.

$$\begin{aligned} min\ & \frac{1}{2}||\mathbf{w}||^2 + \sum_{i=1}^{m} \xi_i \\ & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, m \\ & \xi_i \geq 0 \quad i = 1, \ldots, m \\ & \mathbf{w} \in \mathbb{R}^n \\ & b \in \mathbb{R} \end{aligned} \tag{4.7}$$

As we can see, the kernel function does not appear in the usual formulation $K(\mathbf{x}_i, \mathbf{x}_j)$, where

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{m} (\mathbf{x}_i)_k \cdot (\mathbf{x}_j)_k$$

but instead, it is intrinsic in the use of the hyperplane formulation

$$\mathbf{w} = \sum_{i=1}^{m} \lambda_i y_i \mathbf{x}_i$$

4.2.2 SVM_K-lin_wINT

This model is similar to the one proposed in section 4.2.1, but here we also impose $\mathbf{w}$ to be an integer vector.

$$\begin{aligned} min\ & \frac{1}{2 \cdot S^2}||\mathbf{w}||^2 + \sum_{i=1}^{m} \xi_i \\ & y_i(\frac{1}{S}\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, m \\ & \xi_i \geq 0 \quad i = 1, \ldots, m \\ & \mathbf{w} \in \mathbb{Z}^n \\ & b \in \mathbb{R} \end{aligned} \tag{4.8}$$

where $S \in \{1, 10, 100, 1000\}$ is a scaling factor.

With this model we want to limit the possible values to be assumed by $\mathbf{w}$ vector's components.

Our guess here is that we risk much more overfitting permitting $\mathbf{w}$ to belong to $\mathbb{R}^n$ instead of imposing $\mathbf{w}$ (or $10\mathbf{w}$, or $100\mathbf{w}$, or $1000\mathbf{w}$) to belong to $\mathbb{Z}^n$.

Chapter 5

Experimental tests and results

5.1 Real world datasets

All the datasets used in our experimentation were downloaded from *UCI Machine Learning Repository* [21].

The UCI Machine Learning Repository is a collection of databases, domain theories, and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. The archive was created as an ftp archive in 1987 by David Aha and fellow graduate students at UC Irvine. Since then, it has been widely used by students, educators, and researchers all over the world as a primary source of machine learning data sets.

We decided to test our approaches on eleven datasets, the same ones that also J.P. Brooks used in his work about classification [22].

The chosen datasets contain numerical (real or integer) and categorical (numerical and literal) attributes, and just two possible classes for each item. In other words they are perfectly suitable for binary classification through SVM and other similar approaches.

In table 5.1 we have a summary description of the datasets used in the experimental tests: their name from the repository, their label (used by Brooks), number of instances, and number of features.

5.1.1 Preprocessing on raw data

Data from UCI Repository is not immediately suitable for testing our classification methods.

As Brooks suggests, before their use data has to be properly preprocessed and normalized.

[1]Before-after preprocessing and normalization of the dataset.

Original dataset name [21]	**Mnemonical dataset name**	**Characteristics[1]**	
		# of istances	**# of attributes**
Adult	Adult	30157	105
Statlog (Australian Credit Approval)	Australian	690	14-41
Connectionist Bench (Sonar, Mines vs Rocks)	Sonar	208	60
Pima Indians Diabetes	Pima	769-768	8
Statlog (German Credit Data)	German	1000	24
Statlog (Heart)	Heart	270	13-23
Ionosphere	Ionosphere	351	34-33
Liver Disorders	Bupa	345	6
Breast Cancer Wisconsin (Diagnostic)	Wdbc_Mangasarian_and_Wolberg	568	30
Breast Cancer Wisconsin (Original)	Breast_Mangasarian_and_Wolberg	699-683	9
Breast Cancer Wisconsin (Prognostic)	Wdpc_Mangasarian_and_Wolberg	198-194	33

Table 5.1: Datasets: names, number of instances, number of attributes.

In the following we describe the procedure to be performed on raw data to prepare the input to our classification procedures.

Removing missing values

If there are items with one or more missing values, those items are removed from the relative dataset.

Most of the times, missing attributes are indicated, in the UCI Repository, with a '?', or a '-'.

Converting categorical attributes

If an attribute is of categorical type, that can assume k possible values, it is replaced by k attributes which can attain just 2 possible values (generally 0 or 1).

Example:
If there is an attribute *size*, that has values $\in \{small, medium, large\}$, it is replaced by 3 attributes: *size_1*, *size_2* and *size_3* that have all values $\in \{0, 1\}$.

$size \in \{small, medium, large\}$

...	**size**	...
...	large	...
...	...	...

...	**size$_1$**	**size$_2$**	**size$_3$**	...
...	0	0	1	...
...	...	...	...	...

Converting literal attributes

If an attribute is a literal one, that can attain literal values within a finite set of words of cardinality k, it is conceptually treated as a categorical attribute, so it is replaced by k binary attributes.

Calculating mean value and standard deviation

For each attribute in the dataset, mean value and standard deviation are computed on the training set.

Normalizing the dataset values

First of all, attributes that have standard deviation that is zero are removed from the relative datasets.

Secondly, each attribute value is normalized by subtracting the mean value and dividing by the standard deviation, both calculated, as seen above, on the training set items only.

5.1.2 Experimental choices for data training

In order to perform our experiments, each dataset is randomly partitioned in such a way that the 70% of the items belong to the training set, and the remaining 30% to the test set. This last part of the dataset is taken away at the beginning, and is never used during the whole training procedure: it is used just at the end, during the computation of the final accuracy (the accuracy on the test set) that permits us to evaluate the goodness of the considered approach.

The training part of the dataset is, in turn, divided into 5 parts to perform 5-fold cross validation. This procedure is necessary to choose the optimal parameters to train the system and use the classifier on the test set.

Since our experimental tests require the subsequent resolution of several optimization problems, some of them require a resolution time of about twenty minutes, we decided to restrict in most cases out analysis to nine of the eleven datasets that have less then 800 items. So, for the experimentations described in section 4, we excluded the use of “Adult” and “German” datasets.

5.1.3 Availability of the data on the web

As we said previously, the datasets used, which where taken from the UCI Machine Learning Repository, contain raw data that are inhomogeneous.

Since the programming effort to create a proper parser for each dataset, in order to extract the ordered non zero features and the classes, was not trivial, we thought that it could be useful to make available on the web the postprocessed datasets.

The format we chose is the SVM^{light} format, described in 6.1.2.2, for not normalized dataset items, with categorical and literal attributes converted to numerical ones.

The eleven datasets used can be found at the link

`www.dei.unipd.it/~fisch/datasetSVM.tar.gz`

altogether in a single compressed folder.

5.2 Comparison between *SVM approach* and *A1B0 approach*

In the following we report an analysis about the advantage, in term of accuracy, of a classification based on the classic *SVM approach* (with Gaussian kernel) without any kind of training procedure, just applying the final classifier ($sign[\sum_{i=1}^{m} y_i \lambda_i K(x, x_i) + b]$ – with $\lambda_i = 1\ \forall i$, and $b = 0$).

5.2.1 *A1B0 approach*

This approach just sets

$$\lambda_i = 1 \quad \forall i = 1, \dots, m$$
$$b = 0$$

so that the classifier becomes:

$$sign\left[\sum_{i=1}^{m} y_i K(x_i, x_j)\right] \tag{5.1}$$

where

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2}$$

We want to remark the fact that the use of *A1B0 approach* allows to realize a classification without performing any kind of training procedure on data. For this reason, *A1B0 approach* is much less computationally expensive than *SVM approach*, and also not subject to overtuning.

5.2.2 Parameters setting: $\gamma = 0.1$

SVM approach

In the comparison in analysis only one parameter is free and can be set offline. Namely parameter γ in the Gaussian kernel.

In our experimentation, according to [22], we performed our tests on just few values of γ, i.e.

$$\gamma \in \{0.1, 1, 10, 100, 1000\}$$

As we could experiment in the very numerous tests, in more than the 90% of the cases, the highest accuracies are obtained using $\gamma = 0.1$.

That is the reason why we decided to restrict the interest of this comparison to the use of this value for γ.

For the value of parameter C, we decided to leave the choice to the software used for the computation of the 50 accuracies of *SVM approach*, that is to SVM^{light}.

The default value used is [20]

$$C = [avg.\ x \cdot x]^{-1}$$

that is, the reciprocal of the average of the feature vectors inner products is the value assigned to C.

5.2.3 Organization of the tests

First of all we have to sample randomly each datasets, in order to obtain what we call an *instance* of a dataset.

A single instance of a dataset consists in the random division of the dataset itself in training set (portion of $\frac{7}{10}$ of the dataset) and complementary test set.

We produce 50 different instances for each dataset.

Then, in parallel, we obtain, for each of the aforesaid instances:

- The accuracy with *SVM approach*;
- The accuracy with *A1B0 approach*.

We produced 50 couples of accuracies, where the first element of the couple is obtained with *SVM approach* and the second with *A1B0 approach*.

We can consider each couple as a couple of "measurements" produced by two different "instruments" on the same data – in our case the instrument is the classification procedure used.

We want to:

- Understand, with a rigorous analysis, if the two instruments have a statistically significative difference, or if they are statistically comparable.
- Perform a statistical analysis and comparison of the two methods.

The first evaluation can be done through the use of a statistical method known as *Wilcoxon test*, described in detail in Appendix B.

The second aim can be reached by computing some statistical indicators like mean and standard deviation of the accuracies given by *SVM approach* and by *A1B0 approach*, percentage of *SVM approach* and *A1B0 approach* wins, mean and standard deviation of $\mathbf{\Lambda}$ parameters sparsity for *SVM approach*. These indicators enable us to understand if the use of a SVM is really always useful and advantageous in place of the use of a much simpler and computationally inexpensive method like the classifier in equation (5.1).

5.3 Experimental results for tests described in section 5.2

In this section we will give a global evaluation of the results of the tests described in section 5.2.

5.3.1 Statistical significance of the difference between the two methods

As previously said, we used Wilcoxon test (presented and described in Appendix B, to evaluate if there is a statistically significative difference between *SVM approach* and *A1B0 approach* for the classification of the same test set items.

We perform the test using the statistical software `R`; for details about its installation and use see 6.1.4.

In the following the evaluation of the results and some considerations about them.

Dataset	**p-value**
Adult	< 0.001
Australian	0.0051
Breast Mangasarian and Wolberg	< 0.001
Bupa	< 0.001
German	< 0.001
Heart	< 0.001
Ionosphere	< 0.001
Pima	< 0.001
Sonar	< 0.001
Wdbc Mansagarian and Wolberg	< 0.001
Wpbc Mangasarian and Wolberg	0.0038

Table 5.2: p-values for comparison between *SVM approach* and *A1B0 approach*.

In table 5.3.1 we can see the p-values obtained comparing, through the use of Wilcoxon test technique, 50 couples of measurements, computed respectively with *SVM approach* and *A1B0 approach*.

As we can see, for all the eleven datasets used in our experimentation, we have

$$p - value < 0.01$$

That is, we can state that:

> *SVM approach* is statistically different from *A1B0 approach* with a level of confidence greater than the 99%.

This result could not seem to be very interesting. Considering all the theory and the scientific research about all the possible settings (parameters settings, kernel choices, ...) of Support Vector Machines, we could imagine that *SVM approach* would be always preferable, in terms of accuracy gain, to *A1B0 approach*.

But the results of a statistical analysis and comparison between the two approaches are rather surprising.

5.3.2 Some statistical considerations about the comparison of the two methods

In the following we will evaluate the results for the tests in analysis at two different levels:

- At the level of each single dataset;
- At a high global level comprehensive of all the used datasets.

For the results in detail, at the level of each of the 50 instances taken from each of the eleven datasets, we refer to Appendix A.

5.3.2.1 Datasets level

A global vision of those results at the level of each dataset is given in table 5.3[2].

Dataset	μ_{SVM}	μ_{A1B0}	σ_{SVM}	σ_{A1B0}	SVM>	A1B0>	μ_{spars}	σ_{spars}
Adult	81.79	75.11	0.39	0.38	100.00	0.00	35.99	0.28
Australian	82.78	83.69	2.74	2.51	30.00	70.00	15.51	0.42
Breast Mangasarian and Wolberg	96.84	90.27	0.91	1.97	100.00	0.00	85.15	1.08
Bupa	70.52	58.73	3.90	4.11	98.00	0.00	20.51	2.25
German	72.03	70.03	2.48	2.36	100.00	0.00	23.03	1.56
Heart	79.80	81.78	3.67	4.04	28.00	62.00	21.52	2.12
Ionosphere	93.96	73.28	2.14	4.59	100.00	0.00	41.12	1.99
Pima	75.80	65.83	2.20	3.13	100.00	0.00	42.82	1.86
Sonar	59.59	83.49	7.06	5.15	0.00	100.00	0.61	0.50
Wdbc Mangasarian and Wolberg	95.49	92.68	1.88	2.51	86.00	12.00	55.09	1.04
Wpbc Mangasarian and Wolberg	75.09	75.93	5.06	5.04	16.00	48.00	9.65	2.44

Table 5.3: Results for each dataset for comparison SVM vs A1B0.

Where:

- μ_{SVM} is the mean calculated on the accuracies obtained from *SVM approach*;
- μ_{A1B0} is the mean calculated on the accuracies obtained from *A1B0 approach*;
- σ_{SVM} is the standard deviation calculated on the accuracies obtained from *SVM approach*;

[2]In yellow we marked where the *SVM approach* wins, in green where the *A1B0 approach* wins.

- σ_{A1B0} is the standard deviation calculated on the accuracies obtained from *A1B0 approach*;
- SVM> is the percentage of wins of *SVM approach* (percentage of times the accuracy obtained with *SVM approach* is strictly greater than the one obtained with *A1B0 approach* on the same instance);
- A1B0> is the percentage of wins of *A1B0 approach* (percentage of times the accuracy obtained with *A1B0 approach* is strictly greater than the one obtained with *SVM approach* on the same instance);
- μ_{spars} is the average sparsity (number of zero component of the vector $\mathbf{\Lambda}$) of *SVM approach*;
- σ_{spars} is the sparsity standard deviation of *SVM approach.*

As we can see, differently from our expectations, the *SVM approach* does not lead always to produce higher accuracies than *A1B0 approach.* Actually, in four cases out of eleven, the simpler and computationally inexpensive *A1B0 approach* gives better accuracies than the competitor.

Another interesting observation concerns the values computed for the sparsity of *SVM approach.* The claim of SVM theory is that its approach enables to determine $\mathbf{\Lambda}$ vectors characterized by a high sparsity. This property is quite important and useful because it ensures to decrease the overfitting effect in classification: sparse $\mathbf{\Lambda}$ vectors permits to identify a few points (the so-called *support vectors*) of the training set that really give useful information for the classification of new points (those belonging to the test set), while removing all the other points that, in the final process of classification, could give a misleading information (for example, because they are outliers in the training points system).

Recalling that our analysis is restricted to the use of the Gaussian kernel, with parameter $\gamma = 0.1$, we can observe that actually, except for dataset *Breast Mangasarian and Wolberg*, sparsity of *SVM approach* is not so relevant.

Furthermore, we can observe that, even in the rare cases where the mean sparsity reaches the 50%, this does not have a relevant impact that enables to realize a considerable gain in term of accuracy. In addiction, that these values for sparsity do not affect the use of the SVM with a really sensible reduction of memory use and overall speed of the SVM computations.

5.3.2.2 Higher global level

For an overview of the results at a higher level, we can consider altogether the eleven datasets; the related indicators are given in table 5.4.

where, ad before:

μ_{SVM}	μ_{A1B0}	σ_{SVM}	σ_{A1B0}	SVM>	A1B0>	μ_{spars}	σ_{spars}
83.24	77.35	2.95	3.25	68.91	26.55	31.9	1.41

Table 5.4: Global results for comparison SVM vs A1B0.

- μ_{SVM} is the mean calculated on the accuracies obtained from *SVM approach*;
- μ_{A1B0} is the mean calculated on the accuracies obtained from *A1B0 approach*;
- σ_{SVM} is the standard deviation calculated on the accuracies obtained from *SVM approach*;
- σ_{A1B0} is the standard deviation calculated on the accuracies obtained from *A1B0 approach*;
- SVM> is the percentage of wins of *SVM approach* (percentage of times the accuracy obtained with *SVM approach* is strictly greater than the one obtained with *A1B0 approach* on the same instance);
- A1B0> is the percentage of wins of *A1B0 approach* (percentage of times the accuracy obtained with *A1B0 approach* is strictly greater than the one obtained with *SVM approach* on the same instance);
- μ_{spars} is the mean calculated on the sparsity (number of zero component of the vector $\mathbf{\Lambda}$) of *SVM approach*;
- σ_{spars} is the sparsity standard deviation of *SVM approach.*

Before any analysis of those data, it is fundamental to make an important consideration. The percentage results in table 5.4 are much less significative than the results in table 5.3, because the first give overall statistical information about data whose nature is "clustered". As a matter of fact, considering altogether the eleven datasets, we deal with data showing a quite high similarity within a single dataset, and a quite high dissimilarity between a dataset and another one. In other words, in this case, data are highly inhomogeneous.

However, as we can see, using *SVM approach* (that implies to solve a computationally expensive quadratic programming model), we can get an accuracy advantage just of the 5.9% over the use of a much simpler approach based on the classifier in equation (5.1).

Furthermore, the simpler approach is not always worse than the first one: about 27% times *A1B0 approach* wins giving higher accuracies, and 4-5% times it matches the SVM one.

5.4 Experimental results for tests described in chapter 4

Results are obtained through a preliminary 5-fold cross validation on the training set items, performed in order to get the optimum parameters setting. These parameters will be used to train a final time the system in order to determinate the optimum parameters to classify the test set items, obtaining in such a way the final accuracy.

The aforesaid procedure is then repeated three times and the final percentage accuracy is the mean value of the three accuracies obtained at the end of the three iterations.

For all the methods that involve the use of Gaussian kernel the parameters settings that have been tested are the 25 combinations of five values for the parameters γ and C, as suggested in [22]

$$\gamma \in \{0.1, 1, 10, 100, 1000\}$$

$$C \in \{0.01, 0.1, 1, 10, 100\}$$

The comparison terms for the evaluation of our approaches are the accuracies obtained using, in the procedure above, the classic quadratic programming model with Gaussian kernel.

We give in table 5.5, the average (out of 3) accuracy.

Dataset	**Final accuracy**
Australian	81.8
Breast Mangasarian and Wolberg	96.8
Bupa	67.3
Heart	77.0
Ionosphere	93.7
Pima	74.3
Sonar	82.0
Wdbc Mangasarian and Wolberg	95.5
Wpbc Mangasarian and Wolberg	76.3

Table 5.5: Accuracy with *SVM approach.*

For all the methods that involve the use of the linear kernel, we tested five possible values for parameter C

$$C \in \{0.01, 0.1, 1, 10, 100\}$$

5.4.1 Results with Gaussian kernel

In the following we report the results we got training the system with classifiers obtained by the alternative mathematical models previously described, all using Gaussian kernel. We recall that the results in tables 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12 have to be compared with the ones in table 5.5.

Preliminary considerations

- Small values of γ, in general, give the highest accuracies (in our tests, optimum parameters couples involve, in the 90% of the cases, by the value $\gamma = 0.1$). In other words, high values of γ give scarce accuracies. Recalling the signal analogy explained in figure 3.8, we have that the kernel function can be conceptually associated to the power shape of the signal irradiated by each point in the space around. A big value of γ implies that the kernel function acts like an impulse: it is strong near the item point and much lower in the immediate vicinity of the point itself. In this respect, a kernel function similar to an impulse (Dirac delta), is associated to a classifier that is particularly able to categorize with high accuracy a known point, that is, a point of the training set, instead of a new and unknown point belonging to the test set.

- The values assumed by λ_i's parameters, that we obtained from the resolution of the classical quadratic programming problem with Gaussian kernel, differently from what we imagined, have lots of decimal places, that is, tend to be very fitting to items set in training set.

- Accuracies estimated during the repeated procedure of validation, as we could imagine, are higher than the ones got during the final classification.

The analysis of the behaviour of the system confirm the overfitting phenomenon plays a central role in the procedure.

The above considerations prompted us to plan, implement and test new alternative approaches that try to answer to the question "how can we try to contrast the overfitting effect?"

We undertook the following possibilities:

- Trying to impose the choice of variables λ_i's and b that have less degrees of freedom than in the case of the classical problem – *SVM_aINT model*;

- Trying to impose that, during the validation phase, also a test procedure is realized, in such a way to choose $\mathbf{\Lambda}$ and b parameters whose validity is already tested on a items subset different from the training one – *SVM_TV models*;

- Allowing the system for a certain number of misclassifications in the training set, hoping that this makes the model less fitting to the specific training set and produce more appropriated variables to be used by the final classifier – *SVM_Mixed* and *MixedMu models*.

Final considerations

The goodness of a classifier depends on:

1. The type of the chosen kernel;
2. The validity of the training model;
3. The validation procedure.

Now, we understood that the Gaussian kernel is a good and complete classifier by itself, that is: it provides reasonable accuracies also without taking care to the other two factors just enumerated.

Our claim is that the use of Gaussian kernel reduces the importance of the optimization problem, or makes it even counter-productive in some cases (because of overfitting).

To further prove this thesis about the role of Gaussian kernel, we performed the same tests for the evaluation of the superiority, in terms of accuracy, of *SVM approach* towards *A1B0 approach* (see section 5.3), but this time using linear kernel. This way, we could evaluate the behaviour of the linear kernel as a classifier by itself, compared to the behaviour of the overall *SVM approach* that uses the same linear kernel to perform the final classification over the test set.

We observed that linear kernel is not a so good classifier by itself as Gaussian kernel is. Overall results, for tests performed on 50 instances for each of the eleven datasets, are given in table 5.6. As we can see, the superiority of *SVM approach* towards *A1B0 approach*, that for Gaussian kernel is 5.9%, for linear kernel increases to 7.4%.

μ_{SVM}	μ_{A1B0}	σ_{SVM}	σ_{A1B0}	SVM>	A1B0>	μ_{spars}	σ_{spars}
82.18	74.80	1.19	3.12	79.45	12.38	30.80	1.52

Table 5.6: Global results for comparison SVM vs A1B0 with linear kernel.

SVM_aINT

Dataset	**value of R**	**Final accuracy**
Australian	1	81.5
	10	79.7
	100	77.6
Breast Mangasarian and Wolberg	1	96.9
	10	96.2
	100	96.2
Bupa	1	68.9
	10	67.6
	100	67.0
Heart	1	78.6
	10	77.4
	100	76.7
Ionosphere	1	94.0
	10	93.4
	100	93.7
Pima	1	73.0
	10	74.3
	100	73.7
Sonar	1	78.3
	10	79.4
	100	81.5
Wdbc Mangasarian and Wolberg	1	97.5
	10	96.3
	100	96.3
Wpbc Mangasarian and Wolberg	1	76.3
	10	76.8
	100	76.8

Table 5.7: Accuracy with *SVM_aINT approach.*

SVM_TV_aINT

Dataset	value of R	Final accuracy
Australian	1	77.0
	10	77.6
	100	77.0
	1000	77.6
Breast Mangasarian and Wolberg	1	97.1
	10	96.7
	100	96.1
	1000	96.7
Bupa	1	60.0
	10	60.2
	100	60.2
	1000	58.8
Heart	1	74.1
	10	77.0
	100	76.7
	1000	74.1
Ionosphere	1	92.8
	10	92.8
	100	94.0
	1000	90.7
Pima	1	67.6
	10	68.4
	100	69.0
	1000	68.4
Sonar	1	77.3
	10	76.7
	100	77.3
	1000	80.4
Wdbc Mangasarian and Wolberg	1	93.4
	10	92.8
	100	93.4
	1000	93.6
Wpbc Mangasarian and Wolberg	1	76.8
	10	73.6
	100	73.6
	1000	73.6

Table 5.8: Accuracy with *SVM_TV_aINT approach.*

SVM_TV

Dataset	**Final accuracy**
Australian	79.9
Breast Mangasarian and Wolberg	96.1
Bupa	66.4
Heart	74.9
Ionosphere	93.1
Pima	71.7
Sonar	80.4
Wdbc Mangasarian and Wolberg	92.2
Wpbc Mangasarian and Wolberg	76.8

Table 5.9: Accuracy with *SVM_TV approach.*

SVM_Mixed

Dataset	**C**	**Final accuracy**
Australian	1	79.8
	10	80.5
Breast Mangasarian and Wolberg	-	n.d.[3]
Bupa	1	63.8
	10	64.1
Heart	1	76.7
	10	77.0
Ionosphere	1	87.7
	10	87.7
Pima	1	66.2
	10	65.8
Sonar	1	83.6
	10	85.7
Wdbc Mangasarian and Wolberg	1	97.5
	10	97.5
Wpbc Mangasarian and Wolberg	1	77.4
	10	76.7

Table 5.10: Accuracy with *SVM_Mixed approach.*

[3]CPLEX error in solving: `Q matrix non semi-definite positive because of numerical issues.`

[4]CPLEX error in solving: `Q matrix non semi-definite positive because of numerical issues.`

SVM_aAll

Dataset	**Final accuracy**
Australian	80.7
Breast Mangasarian and Wolberg	94.6
Bupa	67.7
Heart	78.2
Ionosphere	94.3
Pima	70.6
Sonar	83.1
Wdbc Mangasarian and Wolberg	94.3
Wpbc Mangasarian and Wolberg	77.4

Table 5.11: Accuracy with *SVM_aAll_TV approach.*

SVM_MixedMu

Dataset	**C**	**Final accuracy**
Australian	1	78.4
	10	78.4
Breast Mangasarian and Wolberg	-	n.d.[4]
Bupa	1	63.8
	10	64.4
Heart	1	76.3
	10	75.9
Ionosphere	1	82.4
	10	80.7
Pima	1	71.6
	10	72.9
Sonar	1	82.0
	10	82.0
Wdbc Mangasarian and Wolberg	1	94.2
	10	91.7
Wpbc Mangasarian and Wolberg	1	76.3
	10	76.7

Table 5.12: Accuracy with *SVM_MixedMu approach.*

In table 5.13 we have the unified vision of all the results obtained with Gaussian kernel.

Global results

Dataset	**SVM**	**SVM_aINT**	**SVM_TV**	**SVM_TV_aINT**	**SVM_aAll_TV**	**SVM_Mixed**	**SVM_MixedMu**	**A1B0**
Australian	81.8	81.5	79.9	77.0	80.7	80.5	78.4	83.7
Breast	96.8	96.9	96.1	97.1	94.6	–	–	90.3
Bupa	67.3	68.9	66.4	66.0	67.7	64.1	63.8	58.7
Heart	77.0	78.6	74.9	74.1	78.2	77.0	76.3	81.8
Ionosphere	93.7	94.0	93.1	92.8	94.3	87.8	82.4	73.3
Pima	74.3	73.0	71.1	67.6	70.6	65.8	71.6	65.8
Sonar	82.0	78.3	80.4	77.3	83.1	85.7	82.0	83.5
Wdbc	95.5	97.5	92.2	93.4	94.3	97.5	94.2	92.7
Wpbc	76.3	76.8	76.8	76.7	77.4	76.7	76.3	75.9

Table 5.13: All obtained accuracies, with Gaussian kernel.

5.4.2 Results with linear kernel

In the following we report the results with our alternative classifiers using the linear kernel.

Results in table 5.15 are thought to be compared to the ones in table 5.14.

Our attempt here was motivated by the first results obtained from the evaluation of our *SVM_K-lin_wINT approach* on the training set items, conduct at the very beginning without the use of any validation procedure. The aforesaid results are reported in Appendix C.

Dataset	**Final accuracy**
Australian	82.9
Breast Mangasarian and Wolberg	97.1
Bupa	69.6
Heart	81.3
Ionosphere	89.6
Pima	74.5
Sonar	76.2
Wdbc Mangasarian and Wolberg	97.5
Wpbc Mangasarian and Wolberg	78.0

Table 5.14: Accuracy with *SVM_K-lin approach.*

5.5 Final comments over the results

An accurate analysis of the experimental results could raise some doubt about the correctness of the accuracy values, for classical SVM with Gaussian kernel, reported in the various tables of this work. The given accuracies are not everywhere identical.

Actually, the results slightly differ because of the procedure used to obtain them: accuracies reported in the present chapter are means of 3 final accuracy values obtained with relative optimal parameters setting; accuracies reported in Appendix A are values obtained varying the training-test set division and using a offline set parameters couple (C, γ); accuracies reported in Appendix C are values obtained testing all the 25 possible parameters couples (C, γ).

Similar considerations can be done for accuracies of classical SVM with linear kernel.

In other words, it is perfectly reasonable that the reported accuracies vary slightly between them.

Dataset	**value of S**	**Final accuracy**
Australian	1	83.9
	10	84.1
	100	83.3
	1000	84.1
Breast Mangasarian and Wolberg	1	96.9
	10	96.4
	100	96.4
	1000	96.4
Bupa	1	62.8
	10	59.6
	100	57.1
	1000	62.8
Heart	1	80.7
	10	77.4
	100	73.7
	1000	77.0
Ionosphere	1	88.1
	10	88.1
	100	87.7
	1000	87.7
Pima	1	71.7
	10	72.4
	100	71.7
	1000	71.7
Sonar	1	72.0
	10	61.9
	100	54.5
	1000	54.5
Wdbc Mangasarian and Wolberg	1	96.5
	10	96.9
	100	96.3
	1000	96.3
Wpbc Mangasarian and Wolberg	1	76.3
	10	76.8
	100	75.7
	1000	75.7

Table 5.15: Accuracy with *SVM_K-lin_wINT approach*.

Chapter 6

Software

6.1 Used software

6.1.1 CPLEX

IBM ILOG CPLEX Optimization Studio (often informally referred to simply as CPLEX) is an optimization software package.

The CPLEX Optimizer was named after the simplex method as implemented in the C programming language, although today it provides additional methods for mathematical programming and offers interfaces other than just C. It was originally developed by Robert E. Bixby and was offered commercially starting in 1988 by CPLEX Optimization Inc., which was acquired by ILOG in 1997; ILOG was subsequently acquired by IBM in January 2009. CPLEX continues to be actively developed under IBM [3].

CPLEX represents one of the most efficient software application, among those we can find nowadays, created to solve LP and mixed-integer problems.

Precisely, this software enables the user to:

- Solve linear programming problems, even with thousands of variables and constraints, through the simplex algorithm (using either primal or dual variants) and the barrier method;

- Solve integer programming problems, through procedures based on implicit enumeration (i.e. branch and bound);

- Solve quadratic (mixed-integer) problems.

There are two possible ways to interact with CPLEX:

- Writing a linear (integer) programming model in a text file, in the so-called LP format, directly by the user or through a proper software (i.e. GAMS, AMPL, OPL), that is then passed to the solver;

- Using the API of the solver, implementing an interface through source code.

We choose the first alternative, that allows for a better check and debug of the misclassification.

6.1.1.1 Command line code

Since we use the interactive mode, we have to deal with some instructions that enable us to run properly the solver software giving the CPLEX LP file as input (this file format will be described in section 6.1.1.2).

First of all, from the command prompt on a UNIX/Linux system, we have to move to the parent directory that contains the executable file of `cplex`. The run of the program through the command

```
cplex
```

enables us to enter a *CPLEX command prompt*: the bash shell should change to

```
CPLEX>
```

We are now ready to submit the .lp file which contains the PL/PLI problem to be optimized. In the following we will see the basic commands to obtain the optimized solution we need (see [3] for all the other options about the use of the system).

With

```
read filemodel.lp
```

we let the system read the problem.

With

```
opt
```

we let the system start the branch-and-bound/branch-and-cut optimization of the problem.

With

```
write filesolution.sol
```

we let the system write the solution file into the same directory where the program is installed.

With

```
quit
```

we let CPLEX quit and return to the operating mode.

In figure 6.3, an example of interaction with CPLEX from the command shell is given.

Some additional settings

During the optimization phase of CPLEX, some parameters or optimization options can be set.

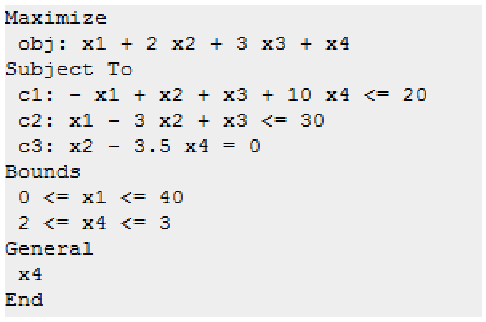
We will give all the relative details when talking about specific optimization runs for the each specific problem.

6.1.1.2 *lp* file format

This file format is one of those accepted in input by CPLEX software.

CPLEX LP format is intended for coding LP/MIP problem data. It is a row-oriented format that assumes the usual formulation of LP/MIP problem.

CPLEX LP file is a plain text file coded using the CPLEX LP format.



```
Maximize
 obj: x1 + 2 x2 + 3 x3 + x4
Subject To
 c1: - x1 + x2 + x3 + 10 x4 <= 20
 c2: x1 - 3 x2 + x3 <= 30
 c3: x2 - 3.5 x4 = 0
Bounds
 0 <= x1 <= 40
 2 <= x4 <= 3
General
 x4
End
```

Figure 6.1: Example of CPLEX LP file.

The fundamental components of a CPLEX LP file are

Objective function definition (mandatory): it must appear at the beginning of the file and must be introduced by the keyword

`MINIMIZE`

or

`MAXIMIZE`

followed by the function to be optimized;

Constraints section (mandatory): it is introduced by the keywords

`SUBJECT TO`

followed by a constraint in one or more rows, where the constant term appears to the right;

Bounds section (optional, when not specified we intend every variable to be nonnegative): it is introduced by the keyword

`BOUNDS`

followed by a bound specification for each variable on a new row;

Variables specification section (optional): it is introduced by the keyword

GENERAL
or
BINARY
or
INTEGER
followed by the list of the relative variables;

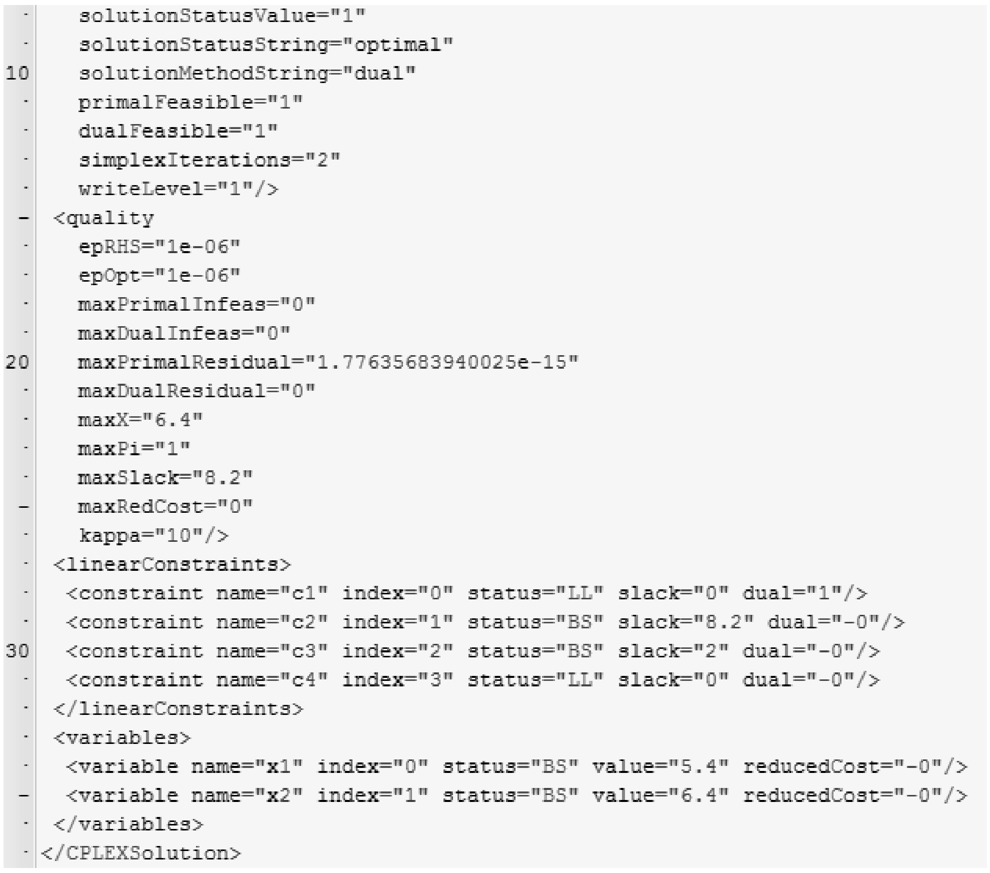
Termination keyword (mandatory):
END.

In figure 6.1, we can see an example of a CPLEX LP file for a small PL model.

6.1.1.3 *.sol* file format

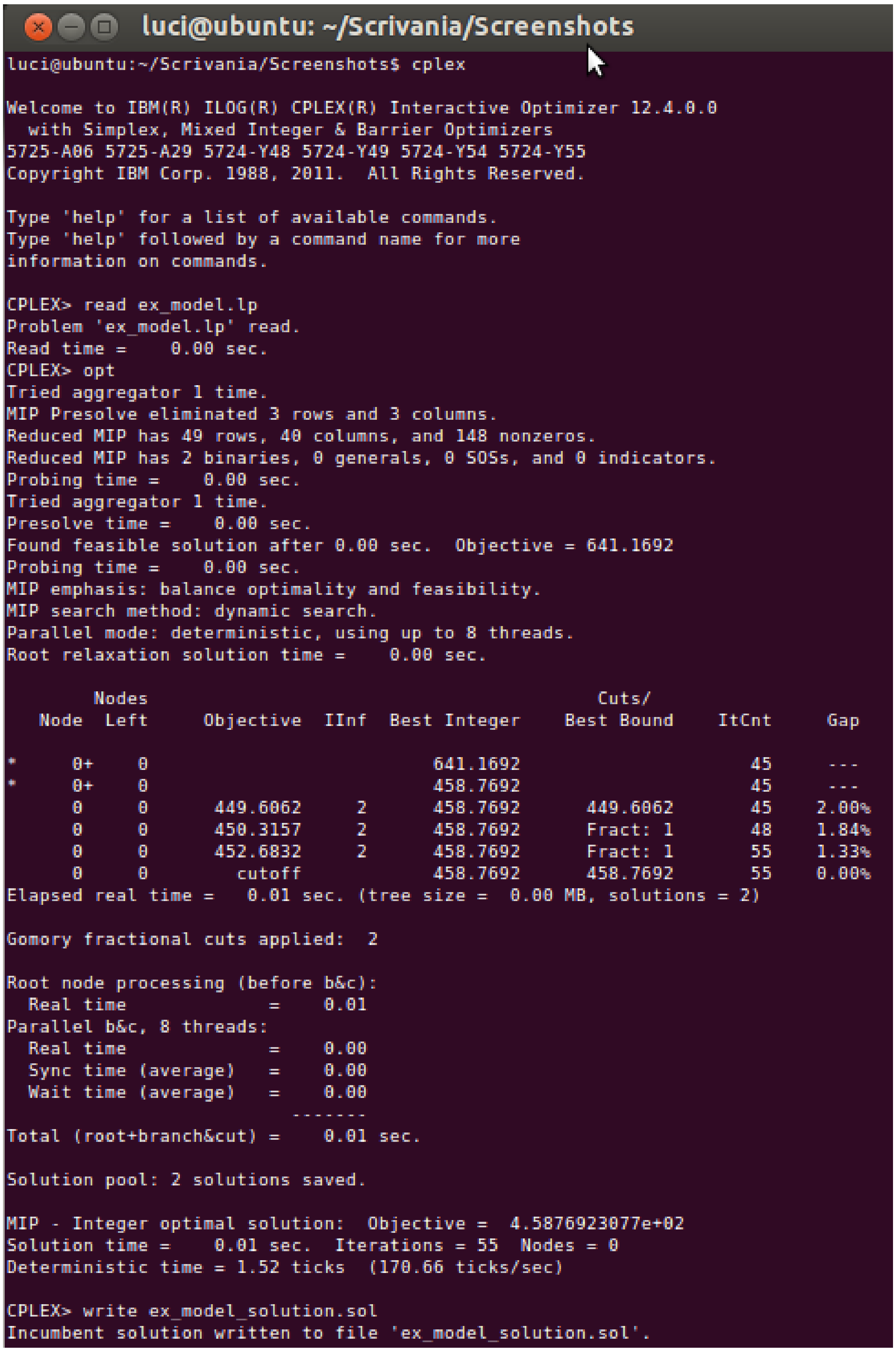
This is the format of files produced in output by CPLEX: they contain lots of information about the solution of the LP model, like variables values at the optimal solution, number of iterations, bounds value to the solution at each iteration, quality of the solution, number of nodes of the solution tree processed and so on ...

In figure 6.2 we can see an example of that kind of file.



```
   solutionStatusValue="1"
   solutionStatusString="optimal"
   solutionMethodString="dual"
   primalFeasible="1"
   dualFeasible="1"
   simplexIterations="2"
   writeLevel="1"/>
 <quality
   epRHS="1e-06"
   epOpt="1e-06"
   maxPrimalInfeas="0"
   maxDualInfeas="0"
   maxPrimalResidual="1.77635683940025e-15"
   maxDualResidual="0"
   maxX="6.4"
   maxPi="1"
   maxSlack="8.2"
   maxRedCost="0"
   kappa="10"/>
 <linearConstraints>
  <constraint name="c1" index="0" status="LL" slack="0" dual="1"/>
  <constraint name="c2" index="1" status="BS" slack="8.2" dual="-0"/>
  <constraint name="c3" index="2" status="BS" slack="2" dual="-0"/>
  <constraint name="c4" index="3" status="LL" slack="0" dual="-0"/>
 </linearConstraints>
 <variables>
  <variable name="x1" index="0" status="BS" value="5.4" reducedCost="-0"/>
  <variable name="x2" index="1" status="BS" value="6.4" reducedCost="-0"/>
 </variables>
</CPLEXSolution>
```

Figure 6.2: Example of CPLEX output file.



luci@ubuntu: ~/Scrivania/Screenshots

```
luci@ubuntu:~/Scrivania/Screenshots$ cplex

Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.4.0.0
  with Simplex, Mixed Integer & Barrier Optimizers
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55
Copyright IBM Corp. 1988, 2011.  All Rights Reserved.

Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.

CPLEX> read ex_model.lp
Problem 'ex_model.lp' read.
Read time =    0.00 sec.
CPLEX> opt
Tried aggregator 1 time.
MIP Presolve eliminated 3 rows and 3 columns.
Reduced MIP has 49 rows, 40 columns, and 148 nonzeros.
Reduced MIP has 2 binaries, 0 generals, 0 SOSs, and 0 indicators.
Probing time =    0.00 sec.
Tried aggregator 1 time.
Presolve time =    0.00 sec.
Found feasible solution after 0.00 sec.  Objective = 641.1692
Probing time =    0.00 sec.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time =    0.00 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                         641.1692                    45     ---
*     0+    0                         458.7692                    45     ---
      0     0      449.6062     2      458.7692      449.6062       45    2.00%
      0     0      450.3157     2      458.7692      Fract: 1       48    1.84%
      0     0      452.6832     2      458.7692      Fract: 1       55    1.33%
      0     0        cutoff            458.7692      458.7692       55    0.00%
Elapsed real time =    0.01 sec. (tree size =  0.00 MB, solutions = 2)

Gomory fractional cuts applied:  2

Root node processing (before b&c):
  Real time             =    0.01
Parallel b&c, 8 threads:
  Real time             =    0.00
  Sync time (average)   =    0.00
  Wait time (average)   =    0.00
                          -------
Total (root+branch&cut) =    0.01 sec.

Solution pool: 2 solutions saved.

MIP - Integer optimal solution:  Objective =  4.5876923077e+02
Solution time =    0.01 sec.  Iterations = 55  Nodes = 0
Deterministic time = 1.52 ticks  (170.66 ticks/sec)

CPLEX> write ex_model_solution.sol
Incumbent solution written to file 'ex_model_solution.sol'.
```

Figure 6.3: Example of use of CPLEX interactive optimizer.

6.1.2 SVM^{light}

SVM^{light} is an implementation of Support Vector Machines in C code, developed by Thorsten Joachims [5].

The main features we are interested in, for this thesis work, are the following:

- fast optimization algorithm
  - working set selection based on steepest feasible descent
  - caching of kernel evaluations
  - use of folding in the linear case
- classification problems resolution
- support for standard kernel functions

SVM^{light} is an implementation of Vapnik's Support Vector Machine for the problem of pattern recognition, for the problem of regression, and for the problem of learning a ranking function. The optimization algorithms used in SVM^{light} are described in [Joachims, 2002a].

SVM^{light} consists of a learning module, called `svm_learn`, and a classification module, called `svm_classify`. The classification module can be used to apply the learning module to new examples.

For the running of the cited modules see section 6.1.2.1.

6.1.2.1 SVM^{light} installation and use

There are several ways of using SVM^{light}. They are all described in [20].

One of those possibilities is to download, in a UNIX/Linux environment, the binary code for both the modules that compose SVM^{light}.

Opening a terminal shell and moving to the directory where we saved the binaries and the input files, we just have to run first the learning module typing

```
svm_learn [options] filetrain filemodel
```

For the whole list of available options see [5].

The options we used in our work are:

`[-t] integer` (kernel type selector – 0:default-linear, 1:polynomial, 2:Gaussian)
`[-g] float` (value of gamma parameter for Gaussian kernel)
`[-c] float` (value of C parameter for quadratic optimization model)
`[-a] string` (file with λ values to be written)

The input file `filetrain` contains the training examples.

The result of `svm_learn` is the model which is learned from the training data in `filetrain`. The model is written to `filemodel`. To make predictions on test examples, `svm_classify` reads this file.

We can run the classification module typing

```
svm_classify [options] filetest filemodel filesolution
```

For the whole list of available options see [5].

The test examples in `filetest` are given in the same format as the training examples.

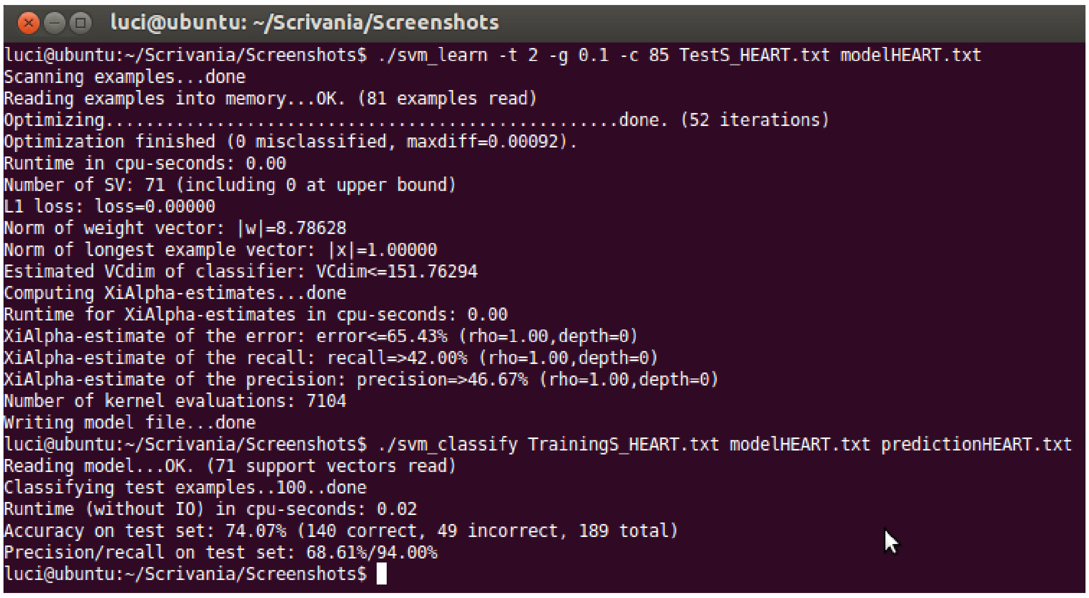
For all test examples in `filetest` the predicted values are written to `filesolution`. There is one line per test example in `filesolution` containing the value of the decision function on that example. This value represents the signed sum computed by the classifier

$$sign\left[\sum_{i=1}^{m} y_i \lambda_i^* K(\mathbf{x}, \mathbf{x}_i) + b^*\right]$$

with the parameters values and the kernel values obtained though the run of `svm_learn` module.

The related class for each test item is the signed of the associate value stored in `filesolution`.

In figure 6.4, an example of interaction with SVM^{light} from the command shell.



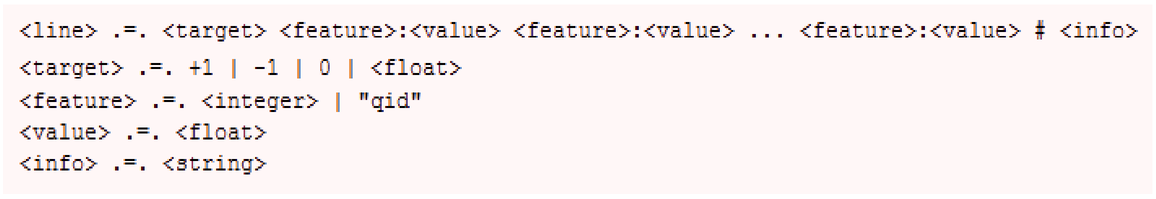
```
luci@ubuntu: ~/Scrivania/Screenshots
luci@ubuntu:~/Scrivania/Screenshots$ ./svm_learn -t 2 -g 0.1 -c 85 TestS_HEART.txt modelHEART.txt
Scanning examples...done
Reading examples into memory...OK. (81 examples read)
Optimizing..................................................done. (52 iterations)
Optimization finished (0 misclassified, maxdiff=0.00092).
Runtime in cpu-seconds: 0.00
Number of SV: 71 (including 0 at upper bound)
L1 loss: loss=0.00000
Norm of weight vector: |w|=8.78628
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=151.76294
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=65.43% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>42.00% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>46.67% (rho=1.00,depth=0)
Number of kernel evaluations: 7104
Writing model file...done
luci@ubuntu:~/Scrivania/Screenshots$ ./svm_classify TrainingS_HEART.txt modelHEART.txt predictionHEART.txt
Reading model...OK. (71 support vectors read)
Classifying test examples..100..done
Runtime (without IO) in cpu-seconds: 0.02
Accuracy on test set: 74.07% (140 correct, 49 incorrect, 189 total)
Precision/recall on test set: 68.61%/94.00%
luci@ubuntu:~/Scrivania/Screenshots$
```

Figure 6.4: Example of use of the modules `svm_learn` and `svm_classify` of SVM^{light}.

6.1.2.2 SVM^{light} file format

This file format is the one that is accepted in input by SVM^{light} software.

Each of the lines in input files represents one training example and is of the format in figure 6.5:



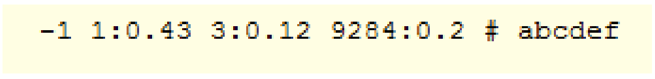
```
<line> .=. <target> <feature>:<value> <feature>:<value> ... <feature>:<value> # <info>
<target> .=. +1 | -1 | 0 | <float>
<feature> .=. <integer> | "qid"
<value> .=. <float>
<info> .=. <string>
```

Figure 6.5: Pattern line of SVM^{light} input files.

Where:

- `target` represents the class the item belongs to
- `feature` represents the ordered index number of the relative feature whose value is given by
- `value` pairs (`feature:value`) where `value` $= 0$ can be skipped

Below in figure 6.6 we can see an example of such a line.



```
-1 1:0.43 3:0.12 9284:0.2 # abcdef
```

Figure 6.6: Example line of SVM^{light} input files.

6.1.3 LIBSVM

In this work we decided also to use another software, that can be directly downloaded from the internet: LIBSVM.

Despite of the fact that we have already used SVM^{light} to realize classification, we decided to have another comparison term, to test the correctness of our results.

We choose LIBSVM for its use simplicity, as we will see in the following.

LIBSVM is an integrated software for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM) developed by Chih-Chung Chang and Chih-Jen Lin [23].

LIBSVM can be simply used through a Python script that makes everything automatic, from data scaling to parameter selection.

LIBSVM provides a simple interface where users can easily link it with their own programs. Some of the features of LIBSVM we are interested in include:

- Cross validation for model selection
- Probability estimates
- Various kernels (including precomputed kernel matrix)

6.1.3.1 LIBSVM installation and use

There are several ways of using LIBSVM. They are all described in [23].

One of those possibilities is to download, in a UNIX/Linux environment, the Python script and the C++ source code that permits to realize classification.

To use this script, it is mandatory to install, always in a UNIX/Linux environment, `Python` and `gnuplot`.

Opening a terminal shell and moving to the directory where we saved the Python script, we just have to run it:

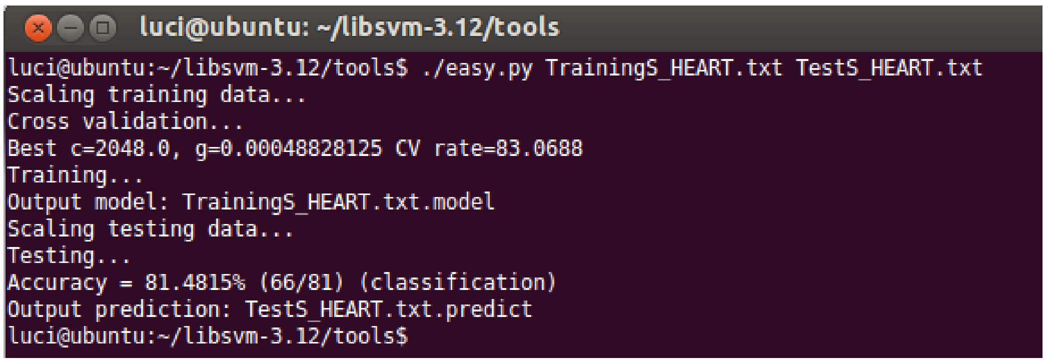
```
./easy filetrain [filetest]
```

For the whole list of available options see [23].

The options we used in our work are:

`[-t] integer` (kernel type selector – 0:default-linear, 1:polynomial, 2:gaussian)
`[-g] float` (value of gamma parameter for gaussian kernel)
`[-c] float` (value of C parameter for quadratic optimization model)
`[-a] string` (file with λ values to be written)

The input file `filetrain` contains the training examples. The input file `filetest` contains the test examples.



```
luci@ubuntu:~/libsvm-3.12/tools$ ./easy.py TrainingS_HEART.txt TestS_HEART.txt
Scaling training data...
Cross validation...
Best c=2048.0, g=0.00048828125 CV rate=83.0688
Training...
Output model: TrainingS_HEART.txt.model
Scaling testing data...
Testing...
Accuracy = 81.4815% (66/81) (classification)
Output prediction: TestS_HEART.txt.predict
luci@ubuntu:~/libsvm-3.12/tools$
```

Figure 6.7: Example of use of the LIBSVM Python script.

In figure 6.7, an example of interaction with LIBSVM from the command shell.

The outputs produced are the accuracies on the testset, in a file named `filetrainoutput.txt`, .log files and a graphical interpretation of classification at the variation of the used parameters (an example in figure 6.8).

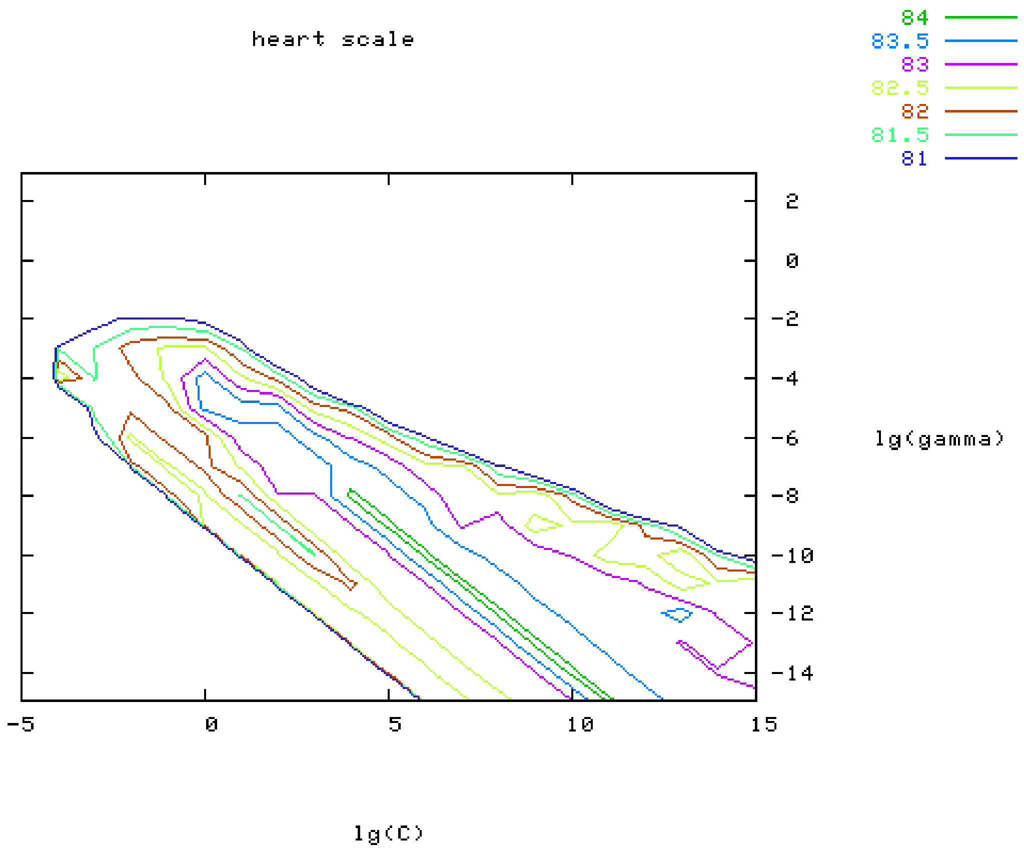


Figure 6.8: Example of a graphical classification representation obtained with LIBSVM.

6.1.3.2 LIBSVM file format

This file format is the one that is accepted in input by LIBSVM software.

This data format is exactly the same accepted in input by SVM^{light} software, so see 6.1.2.2.

6.1.4 R

The most-widely known and used software to perform Wilcoxon test is R.

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulas where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

There are two possible ways of using R:

- as a programming language, that has its own libraries to be linked;
- in interactive way.

6.1.4.1 Command line code

Since we use the interactive alternative, we have to deal with some instructions that enable us to run properly the software to realize the statistical computations we need.

The run of the program through the command

```
R
```

enables us to enter a *R command prompt*: the bash shell should change to

```
R>
```

We are now ready to type three simple commands that perform the Wilcoxon test.

With

a = c(x_1,x_2,...,x_n)

We submit the first array of percentage accuracies.

With

```
b = c(y1,y2,...,yn)
```

We submit the second array of percentage accuracies.

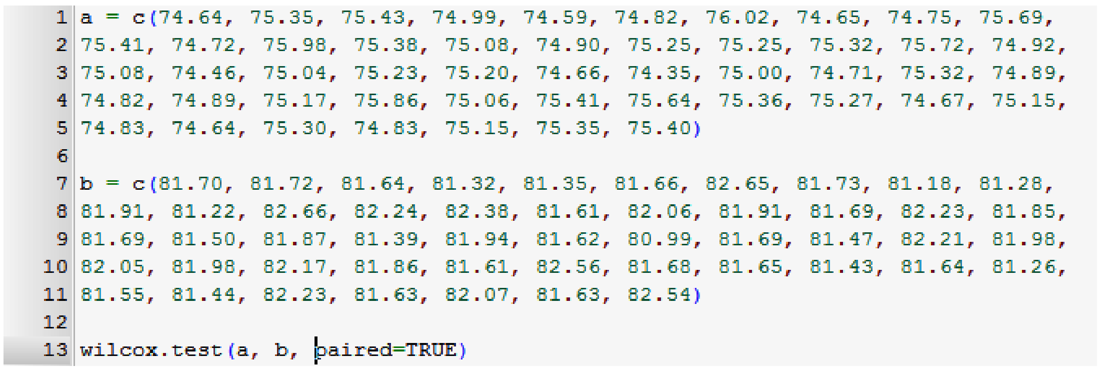
With

```
wilcox.test (a,b, paired=TRUE)
```

We let the system performs the test – defining that the null hypothesis is that the methods compared are equal.

Since R provides also the commands for redirect input and output, we chose this quick alternative.

The previously described commands are written, one for line, in a .txt file, for example the `accur_for_Wilcoxon.txt` in figure 6.9.



```
a = c(74.64, 75.35, 75.43, 74.99, 74.59, 74.82, 76.02, 74.65, 74.75, 75.69,
75.41, 74.72, 75.98, 75.38, 75.08, 74.90, 75.25, 75.25, 75.32, 75.72, 74.92,
75.08, 74.46, 75.04, 75.23, 75.20, 74.66, 74.35, 75.00, 74.71, 75.32, 74.89,
74.82, 74.89, 75.17, 75.86, 75.06, 75.41, 75.64, 75.36, 75.27, 74.67, 75.15,
74.83, 74.64, 75.30, 74.83, 75.15, 75.35, 75.40)

b = c(81.70, 81.72, 81.64, 81.32, 81.35, 81.66, 82.65, 81.73, 81.18, 81.28,
81.91, 81.22, 82.66, 82.24, 82.38, 81.61, 82.06, 81.91, 81.69, 82.23, 81.85,
81.69, 81.50, 81.87, 81.39, 81.94, 81.62, 80.99, 81.69, 81.47, 82.21, 81.98,
82.05, 81.98, 82.17, 81.86, 81.61, 82.56, 81.68, 81.65, 81.43, 81.64, 81.26,
81.55, 81.44, 82.23, 81.63, 82.07, 81.63, 82.54)

wilcox.test(a, b, paired=TRUE)
```

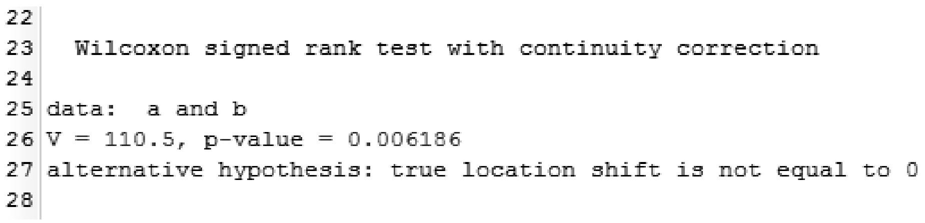
Figure 6.9: Example of LIBSVM input file.

If we want the result of the performed test to be written in a file name, for example, `result_Wilcoxon.txt`, we just have to type:

```
R < accur_for_Wilcoxon.txt --no-save > result_Wilcoxon.txt
```

Where `--no-save` is the default argument (that nevertheless must be declared) that specifies not to save the R workspace at exit.

The output produced contains the lines in figure 6.10.



```

        Wilcoxon signed rank test with continuity correction

data:  a and b
V = 110.5, p-value = 0.006186
alternative hypothesis: true location shift is not equal to 0

```

Figure 6.10: Significative output lines from R software for Wilcoxon test.

6.2 Implemented software

In the following an overview about the software we needed to implement and execute for our tests.

6.2.1 C source code

As we saw in section 6.1, we used software directly downloaded from the web.

The use of that software required the implementation of several programs, written in C code, that enabled us to realize an adequate interface to the software itself.

In figure 6.11, we schematically take a vision of the whole procedure (that integrates the use of implemented and downloaded software) which permits us to simulate the overall operating process of the Support Vector Machines. And also, with some proper changes, of other numerous variants, implemented in this thesis work, of the classical problem.

The starting point of the whole procedure is represented by the **dataset file**, that is the file containing the specific dataset directly downloaded from the UCI Repository [21].

Dataset stored in the aforesaid repository have not got a predefined structure: in general everyone has items and features saved in a different format.

Some have just numerical attributes, but however they can be stored in different ways: an item for each line, with attributes separated by commas, or by hyphens, or by tabs, or dots, and so on. Some other datasets have also categorical attributes: they have to be converted in binary attributes as seen in 5.1. Other datasets contain even literal attributes, that first of all have to be converted in categorical attributes and then to numerical ones.

Parser 1

Parser 1 is an executable that "extracts" the dataset from its original format and structures it in a particular format that is the same for all the datasets.

Between all the possible formats we chose the one that is accepted in input either by SVM^{light} and by LIBSVM (see 6.1.2.2 for details). That way, we could also have the possibility of making a debug of our code in the first phases of our experimentation and also of comparing the final accuracies produced by our approaches with the ones produced by these well known and reliable software.

Beside the formatting of the dataset, *Parser 1*, receiving in input a *seed* for the C method `srand()`, produce also the random division of the dataset in the $\frac{7}{10}$ training and the $\frac{3}{10}$ test set.

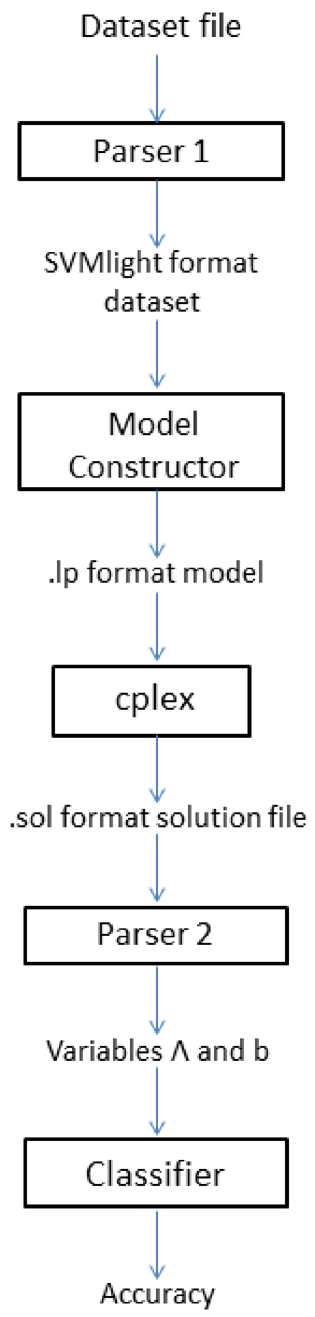


Figure 6.11: Scheme flow for our SVM implementation.

Pseudocode 6.1: Pseudocode for the extraction of n_train different indexes between 0 and n_train-1.

```
init seed s
init array of different indexes el[n_train]
r = rand() % n_train
t_e = 0  //number of taken elements

while (t_e < n_train)
    found = false
    for (i from 0 to t_e-1)
        if (el[i] == r)
            found = true
            break
    if (found == false)
        el[t_e] = r
    t_e++
    r = rand() % n_train
```

Random indexes produced with the procedure in listing 6.1, are placed in ascending order. Then the lines with the corresponding number equal to each index are extracted from the original dataset file and printed in a new file that represents the training set, as described in listing 6.2.

Complementary indexes, at the same time, allow to extract the remaining dataset items, which will form the test set.

Pseudocode 6.2: Pseudocode for the extraction of the training set from the dataset.

```
init training matrix t_m[n_train][# feature]
c_r = 0  //number of the currently read row of the dataset
sc = 0   //index for scanning the array of random ordered indexes
line = read()  //read a line of the dataset

while (c_r < n_dataset)
    if (c_r = el[sc])
        for (each feature k)
            t_m[c_r][k] = feature k of line #c_r
        c_r++
        sc++
    else
        c_r++
    line = read()
```

Parser 1, according to [22], realizes also the normalization of the dataset items, subtracting the mean and dividing by the standard deviation, both calculated on the training set items.

Pseudocode 6.3: Pseudocode for the calculation of mean and standard deviation and for the normalization of the training set.

```
init array of mean values of the attributes mean[#features]

for (i from 0 to #features-1)
```

```
    for (j from 0 to #n_train-1)
        mean[i] = mean[i] + t_m[j][i]
    mean[i] = mean[i]/n_train

init array of standard deviations stdr_dev[#features]

for (i from 0 to #features-1)
    for (j from 0 to #n_train-1)
        stdr_dev[i] = stdr_dev[i] + (t_m[j][i] - mean[i])^2
    stdr_dev[i] = sqrt(stdr_dev[i]/(n_train-1))

for (i from 0 to #features-1)
    for (j from 0 to #n_train-1)
        t_m[i][j] = (t_m[i][j] - mean[i])/stdr_dev[i]
```

In progress, during the experimentations, *Parser 1* has been further complicated in order to realize also the division of the training set in the 5 folds to be used during 5-fold cross validation (the procedure is similar to the one described in pseudocode 6.1 and 6.2).

ModelConstructor

ModelConstructor is the executable that constructs the mathematical model in .lp file format, the one accepted in input by CPLEX (for details see 6.1.1.2).

Obviously lots of model constructors have been implemented because each different mathematical model we tested is formatted by a different code.

The constructor, in order:

- Reads the training set;
- Constructs the relative kernel matrix (pseudocode in listing 6.4);
- Computes the coefficients of the objective function and the bounds;
- Formats the relative .lp model (pseudocode in listing 6.5).

Pseudocode 6.4: Pseudocode for the computation of the kernel matrix.

```
init kernel matrix k[n_train][n_train]
init gamma  //parameter of Gaussian kernel

for (i from 0 to n_train-1)
    for (j from 0 to n_train-1)
        sum = 0
        for (k from 0 to #features-1)
            sum = sum + (t_m[j][k] - t[i][k])^2
        sum = - sum * gamma
        k[i][j] = exp(sum)
```

Pseudocode 6.5: Pseudocode for formatting the .lp file.

```
print to .lp ''MINIMIZE\n''
print to .lp ''OBJ: ''
for (i from 0 to n_train-1)
        print to .lp '' + C x_i''
for (i from 0 to n_train-1)
    for (j from 1 to n_train-1)
        if (i==0)
            if (i==j)
                if (coeff[i][j] > 0)
                    print to .lp (''[%lf a_i^2 '', coeff[i][j])
                else if (coeff[i][j] < 0)
                    c = - coeff[i][j]
                    print to .lp (''[ - %lf a_i^2 '', c)
            else
                if (coeff[i][j] > 0)
                    print to .lp (''+ %lf a_i*a_i '', coeff[i][j])
                else if (coeff[i][j] < 0)
                    c = - coeff[i][j]
                    print to .lp ('' - %lf a_i*a_i '', c)
        else if (i==n_train-1)
            if (i==j)
                if (coeff[i][j] > 0)
                    print to .lp (''+ %lf a_i^2]\2'', coeff[i][j])
                else if (coeff[i][j] < 0)
                    c = - coeff[i][j]
                    print to .lp (''- %lf a_i^2]\2'', c)
        else
            if (i==j)
                if (coeff[i][j] > 0)
                    print to .lp (''+ %lf a_i^2 '', coeff[i][j])
                else if (coeff[i][j] < 0)
                    c = - coeff[i][j]
                    print to .lp (''- %lf a_i^2'', c)

print to .lp ''SUBJECT TO:\n''
for (i from 0 to n_train-1)
    print to .lp (''V%d:\n'', i+1)
    for (j from 0 to n_train-1)
        print to .lp (''+ %lf a_i '', coeff[i][j])
    if (y[i]>0)
        print to .lp (''+ b + x%d >= 1\n'', i+1)
    else
        print to .lp (''+ b + x%d >= 1\n'', i+1)

print to .lp ''BOUNDS:\n''
print to .lp ''b free''

print to .lp ''END''
```

Parser 2

Parser 2 is an executable that “extracts” from .sol file produced in output by CPLEX (for details see 6.1.1.2) just the variables that are used in the classifier, that is, the optimal $\mathbf{\Lambda}^*$ and b^* produced with the optimization software.

Pseudocode 6.6: Pseudocode for the extraction of variables indexes and values from .sol file.

```
to_be_found = ''variables''
line = readline()  //read a line from .sol file
init file_index
init file_values

while (line != NULL)
    search to_be_found in line
    if found
        while (line is not ended)
            search variable name
            if (name == b)
                search variable value and write in file_b
            else if (name = a)
                search variable number and write in file_index
                search variable value and write in file_values
    line = read(line)

init array for variables alpha[n_train]
for (i from 0 to n_train-1)
    ind = read(file_index)
    val = read(file_values)
    alpha[ind-1] = val
```

Classifier

Classifier is an executable that implements the classifier

$$sign\left[\sum_{i=1}^{m} y_i \lambda_i^* K(\mathbf{x}, \mathbf{x}_i) + b^*\right] \tag{6.1}$$

on the test set points, $\mathbf{x}$.

The *Classifier*, in sequence:

- Calculates the kernel matrix (with a procedure similar to that in listing 6.4, but this time using both training and test set items);
- Implements (listing 6.7) the classification procedure described by equation (6.1);
- Compares, for each test set item, the predicted class with the real one, known in advance;
- Based on the number of misclassifications committed, computes the final accuracy on the test set (listing 6.8).

Pseudocode 6.7: Pseudocode for the calculation of the predicted classes for test set items.

```
init the test set item classes y1[n_test]
init the training set item classes y[n_train]
init the kernel matrix k[n_test][n_train]
init the array for each item classification sum[n_test]
init the array for each item predicted class sign[n_test]
```

```

for (i from 0 to n_test-1)
    sum[i] = 0
    for (j from 0 to n_train-1)
        sum[i] = sum[i] + k[i][j]*alpha[j]*y[j]
    sum[i] = sum[i] + b
    if (sum > 0)
        sum[i] = 1
    else
        sum[i] = -1
```

Pseudocode 6.8: Pseudocode for the computation of the accuracy.

```
init the test set item classes y1[n_test]
acc = 0

for (i from 0 to n_test-1)
    if (y1[i] == sign[i])
        acc++

final_accuracy = (acc/n_test)*100
```

6.2.2 Scripts

The tests performed in our experimentation require to repeat the whole procedure several times.

As a matter of fact, we have to remember that every single accuracy obtained in our experimentation is the mean value of three accuracies derived from a whole procedure of 5-fold cross validation (5 iterations per parameters setting, the most of the times we worked combining the simultaneous try of two parameters in a set of 5 values – that is, for each cross validation there are 25 executions, for a total of 125 sequential executions).

What we want to say is that it was necessary to implement proper scripts that enable us to start just a single time the whole procedure and not to lose time compiling and executing several times our codes.

Two examples of scripts, respectively for the realization of the *SVM approach* and for the comparison between *SVM approach* and *A1B0 approach* on the dataset Heart are given in Appendix D.

6.2.3 Hardware specifications

Tests have been performed on a machine Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz, and 4GB RAM.

6.3 Instructions for compilation and execution of C source code

First of all we will describe the structure of directories and subdirectories for saving, compiling and executing all the source code implemented in this thesis project.

Secondly we will give some instructions for the use of the source code.

6.3.1 Directories structure

To compile and execute properly the source code implemented in our work, the following directories structure has to be created:

```
> CODES/
            > File_txt/
                        > Accuracy/
                        > Alpha/
                        > Dataset/
                        > PerSVMlight/
> SCRIPTS/
            > 50_A1B0/
            > 50_SVMlight/
            > SVM_aINT/
            > SVM_aAll_TV/
            > SVM_K-gauss/
            > SVM_K-lin/
            > SVM_K-lin_wINT/
            > SVM_Mixed/
            > SVM_MixedMu/
            > SVM_TV/
            > SVM_TV_aINT/
> OUTPUT/
```

Where

- `50_A1B0/`: refers to the realization of *A1B0 approach*;
- `50_SVMlight/`: refers to the realization of *SVM approach*;

- `SVM_aINT/`: refers to the realization of classification using *a_INT model*;
- `SVM_aAll_TV/`: refers to the realization of classification using *aAll_TV model*;
- `SVM_K-gauss/`: refers to the realization of classification using *K-gauss model*;
- `SVM_K-lin/`: refers to the realization of classification using *K-lin model*;
- `SVM_K-lin_wINT/`: refers to the realization of classification using *K-lin_wINT model*;
- `SVM_Mixed/`: refers to the realization of classification using *Mixed model*;
- `SVM_MixedMu/`: refers to the realization of classification using *MixedMu model*;
- `SVM_TV/`: refers to the realization of classification using *TV model*;
- `SVM_TV_aINT/`: refers to the realization of classification using *TV_a_INT model*;

6.3.2 Where to save files

All the datasets, in the original format from UCI Repository, has to be saved in the directory `Dataset/` as

```
dataset_<DATASET_NAME>.txt
```

In the directory *CODES/* have to be saved:

- C source codes: `*.c` files
- parameters for CPLEX: `*.dat` files

The directory containing the scripts, as we can see, is articulated in a set of subdirectories, one for each of the mathematical models we tested.

There is a specific script, with proper parameters, for each dataset and for each model to be tested[1].

If the script subdirectory is, for example, `SVM_K-lin/`, scripts named

```
throws<DATASET_NAME>_SVM_K-lin.sh
```

have to be saved inside it.

If the subdirectory is `SVM_TV/`, scripts named

```
throws<DATASET_NAME>_SVM_TV.sh
```

have to be saved inside it.

And so forth for all the other scripts to be saved in all the other subdirectories of directory `SCRIPTS`.

[1]A further work to be done could be to unify all the scripts for a model in a single script.

6.3.3 How to make the code work

Source code has been implemented and executed on a Linux environment, so correctness and efficacy of the following instructions for the code execution is guaranteed only for this type of platform.

6.3.3.1 Prerequisites

CPLEX

All the scripts, except the ones in directory `50_SVMlight/` and in directory `50_A1B0/`, requires the previous installation of the interactive CPLEX optimization solver (for details see 6.1.1).

Notice that, reading a script, we can see that CPLEX is called typing

```
cplex
```

instead of

```
./cplex
```

That means that the execute permission for CPLEX has to be previously extended to all.

This can be done typing, by the command prompt after moved to the directory containing its executable

```
chmod a+x cplex
```

SVM*light*

All the scripts in directory `50_SVMlight/` requires the previous installation of SVM*light* software (for details see 6.1.2).

Notice that, reading a script, we can see that module `svm_learn` is called typing

```
svm_learn
```

instead of

```
./svm_learn
```

That means that the execute permission for `svm_learn` has to be previously extended to all.

This can be done typing, by the command prompt after have moved to the directory containing its executable

```
chmod a+x svm_learn
```

The same considerations and consequent procedure above apply to module `svm_classify`.

6.3.3.2 Instructions for scripts runs

The organization of the source code has been thought with the aim of reducing the number of programs to be used, trying to adopt a modular approach in the implementation of the code itself.

Conversely, the organization of the scripts is probably redundant, but permits to have the total control over the specific codes execution.

In the following an example to clarify the use of the scripts.

First of all we have to make all the scripts executable. This can be done, from a command prompt, moving to the directories where we can find the scripts and typing

```
chmod +x <SCRIPT_NAME>
```

If we want to know the final mean accuracy for Ionosphere dataset, produced by the classification approach based on the so-called *TV model*, we just have to move with the command prompt to `SCRIPTS/SVM_TV/` directory and type

```
. ./throwsIONOSPHERE_SVM_TV_validation.sh > outputfile.txt
```

This way, all the outputs produced by the optimization of CPLEX and eventually by the execution of codes are redirected to an output file that can be use for controls and debugs.

The final accuracy will be printed in a file named

```
Accuracy.txt
```

in directory `OUTPUT/`.

Conclusions and future perspectives

This thesis work involved a particularly intense effort and dedication. It represented for the student a challenge in the use and the acquisition of familiarity with new software tools and new programming procedures.

We started our analysis from documentation about the theoretical foundations of the functionality of Support Vector Machines. And we faced a practical experience with SVM software tools like SVM^{light} and LIBSVM.

Afterwards, we first provided the global software implementation (comprehensive of C programs, bash scripts and the use of CPLEX as interactive optimizer) that allowed us to emulate a classic SVM, either with Gaussian or linear kernel. Starting from its use we could perform a detailed investigation (that required several tests) of the behaviour of its variables, which are determined by the resolution of an optimization method and realize the final classification. That enabled us to confirm that the behaviour of the system could suffer for the overfitting phenomenon. Knowing that, we planned and implemented several alternative approaches, all aimed at ensuring a functionality with less overfitting.

We observed, first of all, that the proposed approaches, which are alternative to the classic SVM approach with Gaussian kernel, globally guarantee performances that are equivalent, in terms of accuracy, to the ones produced by SVM approach itself.

Secondly, we understood that the performances of the overall functionality of a SVM depends on three factors: the optimization model, the kernel type and the trial-and-error validation of the parameters setting. In this context our analysis pointed out that the Gaussian kernel tends to act like a good classifier by itself, characteristic that considerably limits the rooms for improvement that we could think to achieve acting on the remaining two factors.

Then we evaluated the entity of the superiority, in terms of accuracy, of the SVM approach towards the one based just on the use of the final SVM classifier. In the last one the variables values are fixed a priori in such a way to consider each point in the system to give the same information for the classification of new items. This evaluation allowed us to validate our thesis about the good nature of the Gaussian kernel as a classifier by itself: we observed an overall loss lower than the 6% of the simplified approach towards the SVM one; besides, we noticed a bigger loss when replicating the same comparison with the use of linear kernel.

Eventually, we terminated our analysis with the investigation of the behaviour of linear kernel. The information collected during the previous experimentations tells us that the use of this type of kernel surely permits more rooms for improvement. The tests on linear kernel, aimed again to the reduction of overfitting, were performed at the end of this thesis work. That was actually just a first attempt, that we could not deepen for reasons of time.

At the end of our work some hints for future researches can be given: we think that the same experimental tests conducted for Gaussian kernel, using alternative mathematical models, can be replicated also for linear kernel with hopeful improvements of the global performances. We suggest also to act with further modifications on the simplified approach based on the use of the classifier without any kind of optimization and validation procedures: with the proper adaptations it could become a valid alternative to the computationally expensive SVM approach, with comparable accuracy but much less computational effort.

Appendix A

Detailed results for tests described in section 5.2

In the following we report the detailed results obtained, for each dataset instance, for the comparison of *SVM approach* towards *A1B0 approach* with Gaussian kernel. The description of the tests is given in section 5.2.

For *SVM approach*, sparsity is defined as the number of zero component of the vector $\mathbf{\Lambda}$; its value is always 0% for *A1B0*.

Adult

Seed	**SVM approach**	**A1B0 approach**	**SVM sparsity**
1	81.70	74.64	35.92
2	81.72	75.35	36.04
3	81.64	75.43	36.09
4	81.32	74.99	36.24
5	81.35	74.59	36.54
6	81.66	74.82	36.04
7	82.65	76.02	35.60
8	81.73	74.65	35.83
9	81.18	74.75	36.16
10	81.28	75.69	35.53
11	81.91	75.41	35.39
12	81.22	74.72	36.04
13	82.66	75.98	35.70
14	82.24	75.38	35.87
15	82.38	75.08	35.48
16	81.61	74.90	36.07
17	82.06	75.25	35.73
18	81.91	75.25	36.28
19	81.69	75.32	35.81
20	82.23	75.72	35.54
21	81.85	74.92	36.38
22	81.69	75.08	36.26
23	81.50	74.46	35.95
24	81.87	75.04	36.32
25	81.39	75.23	36.25
26	81.94	75.20	36.16
27	81.62	74.66	35.97
28	80.99	74.35	36.65
29	81.69	75.00	36.28
30	81.47	74.71	36.32
31	82.21	75.32	35.92
32	81.98	74.89	35.64
33	82.05	74.82	36.00
34	81.98	74.89	35.54
35	82.17	75.17	35.74
36	81.86	75.86	36.05
37	81.61	75.06	36.35
38	82.56	75.41	35.69
39	81.68	75.64	36.02
40	81.65	75.36	36.05
41	81.43	75.27	35.95
42	81.64	74.67	35.98
43	81.26	75.15	36.23
44	81.55	74.83	36.24
45	81.44	74.64	36.24
46	82.23	75.30	36.16
47	81.63	74.83	36.19
48	82.07	75.15	35.62
49	81.63	75.35	35.93
50	82.54	75.40	35.79
μ	81.79	75.11	35.99
σ	0.39	0.38	0.28

Table A.1: Comparison table for dataset Adult.

Australian

Seed	**SVM approach**	**A1B0 approach**	**SVM sparsity**
1	77.78	76.33	16.77
2	83.09	85.99	17.39
3	83.57	79.71	16.77
4	78.26	81.64	16.15
5	82.61	85.02	17.18
6	83.57	84.54	17.60
7	79.71	82.61	16.36
8	85.51	84.54	14.29
9	85.51	81.64	16.98
10	83.57	86.47	14.49
11	78.26	84.06	15.94
12	85.02	86.47	15.11
13	81.16	85.99	12.22
14	85.51	83.57	16.15
15	82.61	84.06	13.04
16	83.09	85.02	15.94
17	82.13	83.57	15.11
18	81.16	82.61	16.56
19	85.02	85.99	15.72
20	76.81	81.64	15.94
21	84.06	85.02	15.32
22	84.54	85.02	15.32
23	80.68	80.19	16.77
24	81.64	83.09	16.98
25	81.64	83.09	15.94
26	81.64	83.57	12.42
27	78.74	82.13	13.87
28	85.02	81.64	14.91
29	85.02	84.06	16.15
30	80.19	78.74	17.39
31	79.71	82.13	15.94
32	79.23	79.71	14.91
33	82.13	82.61	16.56
34	84.06	85.99	12.42
35	85.99	85.51	15.53
36	85.51	86.96	14.29
37	87.44	88.41	15.11
38	76.81	82.61	16.98
39	84.54	85.02	13.04
40	80.19	82.13	14.29
41	85.99	87.44	16.77
42	81.16	79.23	16.36
43	82.13	83.57	12.63
44	84.06	82.61	15.11
45	85.51	86.47	16.15
46	85.02	88.41	13.87
47	84.54	83.09	16.15
48	86.47	83.57	16.98
49	85.51	84.54	15.73
50	85.99	86.47	16.14
μ	82.78	83.69	15.51
σ	2.74	2.51	0.42

Table A.2: Comparison table for dataset Australian.

Breast Mangasarian and Wolberg

Seed	SVM approach	A1B0 approach	SVM sparsity
1	97.07	91.22	85.15
2	97.07	89.76	86.40
3	97.56	93.66	84.10
4	98.05	90.73	84.73
5	96.10	87.80	86.82
6	96.59	88.29	86.19
7	96.59	90.24	84.94
8	97.56	87.80	84.94
9	97.56	89.27	85.78
10	98.05	92.68	82.43
11	96.59	89.27	86.61
12	95.61	86.34	86.40
13	98.05	90.24	83.89
14	97.56	89.76	84.10
15	96.59	89.76	85.98
16	96.59	85.37	85.77
17	97.07	90.24	85.98
18	99.02	93.17	82.22
19	95.61	92.20	86.82
20	97.56	93.66	84.73
21	96.10	91.22	85.77
22	96.59	91.71	85.98
23	97.56	92.20	83.68
24	97.07	88.29	85.36
25	97.07	92.68	84.73
26	97.07	89.76	85.98
27	96.10	92.68	84.52
28	96.59	87.80	85.77
29	95.61	89.76	86.61
30	95.61	87.32	85.98
31	97.56	89.27	83.68
32	97.56	90.24	84.94
33	96.59	88.29	85.77
34	97.56	89.76	85.36
35	96.10	90.24	85.36
36	94.63	90.24	85.15
37	96.10	92.68	85.77
38	98.05	89.27	83.89
39	97.56	92.20	84.31
40	95.61	89.27	87.03
41	98.54	92.20	84.31
42	96.59	94.15	85.77
43	96.10	92.20	84.94
44	97.07	86.83	85.36
45	96.10	89.76	84.52
46	97.07	90.24	85.15
47	95.61	90.24	85.98
48	96.10	90.24	84.10
49	95.61	91.22	84.10
50	98.05	90.24	83.89
μ	96.84	90.27	85.15
σ	0.91	1.97	1.08

Table A.3: Comparison table for dataset Breast Mangasarian and Wolberg.

Bupa

Seed	SVM approach	A1B0 approach	SVM sparsity
1	67.31	53.85	22.41
2	67.31	52.88	23.24
3	73.08	61.54	19.92
4	63.46	54.81	22.82
5	75.00	62.50	20.33
6	64.42	56.73	22.82
7	67.31	50.00	21.16
8	70.19	55.77	19.50
9	72.12	59.62	19.50
10	73.08	54.81	18.67
11	67.31	57.69	22.82
12	74.04	59.62	16.18
13	75.00	60.58	18.67
14	66.35	59.62	22.40
15	70.19	66.35	19.50
16	72.12	58.65	18.67
17	72.12	60.58	20.33
18	69.23	53.85	18.26
19	68.27	58.65	21.16
20	71.15	63.46	21.16
21	68.27	58.65	19.09
22	74.04	61.54	19.92
23	68.27	67.31	21.16
24	72.12	61.54	22.82
25	74.04	60.58	19.50
26	72.12	61.54	17.01
27	68.27	60.58	21.58
28	72.12	56.73	20.75
29	62.50	62.50	24.07
30	67.31	58.65	22.41
31	75.00	54.81	16.60
32	73.08	51.92	19.09
33	75.96	56.73	19.92
34	76.92	58.65	19.50
35	69.23	54.81	23.24
36	59.62	48.08	25.31
37	67.31	56.73	23.65
38	71.15	58.65	21.16
39	72.12	62.50	22.41
40	68.27	62.50	22.82
41	64.42	52.88	24.07
42	76.92	59.62	16.60
43	76.92	63.46	17.43
44	67.31	54.81	22.41
45	75.00	64.42	16.60
46	71.15	56.73	20.33
47	70.19	59.62	20.33
48	73.08	61.54	19.92
49	72.12	64.42	19.92
50	72.12	62.50	16.60
μ	70.52	58.73	20.51
σ	3.90	4.11	2.25

Table A.4: Comparison table for dataset Bupa.

German

Seed	SVM approach	A1B0 approach	SVM sparsity
1	74.00	72.00	23.57
2	76.00	74.33	22.57
3	75.00	71.00	23.29
4	69.33	67.67	25.43
5	72.67	69.67	23.86
6	72.00	70.67	21.86
7	73.00	71.00	23.71
8	72.33	70.67	23.00
9	69.67	67.67	23.71
10	73.00	72.33	24.43
11	73.33	71.00	24.57
12	67.00	64.67	27.57
13	72.67	70.00	23.00
14	74.33	70.67	21.71
15	73.33	71.33	22.86
16	69.67	68.00	21.00
17	76.67	75.67	21.29
18	72.33	71.33	21.71
19	71.67	70.00	23.29
20	71.33	69.00	23.71
21	73.67	70.00	22.14
22	72.33	69.33	23.71
23	70.33	68.00	23.86
24	69.67	69.00	25.14
25	72.33	69.33	22.00
26	70.00	69.00	22.71
27	73.67	70.67	18.71
28	73.00	71.33	22.00
29	69.67	68.00	24.86
30	73.67	71.00	21.14
31	74.33	73.00	22.42
32	72.67	70.33	22.00
33	73.00	69.33	24.00
34	71.67	70.00	24.43
35	69.00	65.33	25.00
36	64.67	63.67	24.86
37	68.67	67.00	22.57
38	70.33	70.00	24.00
39	76.00	74.67	21.14
40	71.67	69.67	22.57
41	68.00	67.67	23.86
42	71.00	70.33	22.00
43	73.67	70.33	22.14
44	72.67	69.33	22.71
45	73.33	72.00	23.86
46	74.33	71.67	20.86
47	76.00	74.00	20.57
48	67.00	66.33	25.43
49	71.33	69.67	23.00
50	74.33	72.67	21.43
μ	72.03	70.03	23.03
σ	2.48	2.36	1.56

Table A.5: Comparison table for dataset German.

Heart

Seed	SVM approach	A1B0 approach	SVM sparsity
1	76.54	81.48	20.11
2	79.01	80.25	21.16
3	72.84	76.54	22.22
4	85.19	87.65	19.05
5	79.01	83.95	20.63
6	79.01	82.72	20.11
7	79.01	81.48	18.52
8	81.48	80.25	18.52
9	75.31	79.01	20.63
10	75.31	77.78	22.75
11	77.78	88.89	19.05
12	86.42	86.42	20.11
13	82.72	82.72	22.75
14	80.25	83.95	23.81
15	75.31	76.54	20.63
16	83.95	90.12	21.16
17	79.01	85.19	20.63
18	82.72	82.72	19.05
19	80.25	76.54	21.16
20	82.72	81.48	19.05
21	77.78	77.78	22.22
22	79.01	77.78	22.22
23	79.01	80.25	23.81
24	82.72	81.48	23.81
25	85.19	86.42	17.99
26	70.37	77.78	25.93
27	80.25	87.65	18.52
28	83.95	85.19	21.69
29	77.78	80.25	20.63
30	82.72	86.42	23.28
31	76.54	76.54	25.93
32	76.54	80.25	20.11
33	76.54	74.07	23.28
34	76.54	77.78	25.40
35	76.54	79.01	21.69
36	77.78	82.72	21.69
37	82.72	81.48	22.75
38	80.25	79.01	21.69
39	81.48	85.19	22.75
40	85.19	81.48	21.69
41	83.95	86.42	20.11
42	87.65	90.12	19.05
43	79.01	81.48	21.16
44	76.54	74.07	23.28
45	77.78	81.48	21.69
46	80.25	82.72	22.22
47	86.42	88.89	16.93
48	76.54	79.01	24.34
49	81.48	80.25	23.81
50	77.78	80.25	25.40
μ	79.80	81.78	21.52
σ	3.67	4.04	2.12

Table A.6: Comparison table for dataset Heart.

Ionosphere

Seed	**SVM approach**	**A1B0 approach**	**SVM sparsity**
1	92.45	80.19	38.78
2	91.51	69.81	40.41
3	96.23	67.92	40.82
4	94.34	71.70	42.45
5	93.40	73.58	41.63
6	90.57	65.09	44.49
7	96.23	79.25	37.55
8	97.17	80.19	40.41
9	96.23	73.58	39.18
10	94.34	72.64	41.63
11	94.34	77.36	40.82
12	98.11	69.81	38.78
13	95.28	65.09	42.45
14	93.40	75.47	40.41
15	92.45	66.98	44.90
16	92.45	72.64	41.63
17	92.45	67.92	46.12
18	96.23	78.30	37.55
19	96.23	74.53	42.63
20	95.28	75.47	41.63
21	93.40	76.42	42.45
22	89.62	76.42	42.45
23	89.62	74.53	41.63
24	96.23	77.36	42.04
25	95.28	72.64	38.37
26	94.34	71.70	39.59
27	92.45	80.19	40.82
28	92.45	70.75	43.67
29	94.34	75.47	41.22
30	92.45	66.98	43.67
31	94.34	74.53	42.04
32	95.28	69.81	40.00
33	94.34	78.30	41.63
34	94.34	76.42	38.78
35	93.40	71.70	40.82
36	95.28	74.53	37.96
37	96.23	72.64	42.04
38	92.45	67.92	42.04
39	92.45	80.19	39.59
40	96.23	71.70	40.41
41	95.28	71.70	41.63
42	92.45	76.42	42.45
43	90.57	78.30	37.96
44	95.28	75.47	41.63
45	98.11	73.58	38.37
46	96.23	82.08	39.18
47	90.57	68.87	44.90
48	94.34	70.75	39.59
49	90.57	61.32	42.04
50	91.51	67.92	42.86
μ	93.96	73.28	41.12
σ	2.14	4.59	1.99

Table A.7: Comparison table for dataset Ionosphere.

Pima

Seed	SVM approach	A1B0 approach	SVM sparsity
1	74.46	67.97	46.18
2	72.73	69.26	44.13
3	75.32	65.37	43.76
4	77.49	68.83	41.34
5	75.32	66.23	42.27
6	76.62	64.07	44.69
7	74.03	64.94	45.44
8	76.19	64.94	45.07
9	76.19	67.10	42.83
10	73.59	61.90	44.51
11	76.19	72.29	40.41
12	80.09	68.40	40.41
13	76.62	70.56	42.64
14	78.35	64.94	42.27
15	76.19	68.40	43.20
16	74.03	60.61	43.02
17	75.32	65.80	42.27
18	71.43	56.28	48.60
19	75.32	67.10	43.95
20	73.16	66.67	40.97
21	72.73	66.23	43.39
22	76.19	67.53	43.20
23	76.19	64.07	40.78
24	78.35	68.83	40.22
25	73.16	62.34	44.32
26	78.35	65.80	42.46
27	74.89	64.50	41.15
28	74.89	59.31	45.44
29	78.35	67.53	41.15
30	76.62	64.50	42.46
31	74.46	62.77	43.58
32	75.32	67.97	42.83
33	78.35	66.23	40.78
34	78.35	70.13	40.41
35	74.89	66.23	43.95
36	78.79	71.86	40.22
37	75.76	64.07	41.15
38	77.92	64.50	43.76
39	78.35	65.80	41.71
40	79.22	64.50	40.60
41	77.92	67.97	43.20
42	74.46	65.37	42.83
43	75.76	71.86	40.60
44	74.89	63.20	44.32
45	76.62	61.90	41.71
46	77.49	65.80	42.27
47	68.40	61.90	47.11
48	76.62	64.94	41.90
49	74.46	66.67	42.09
50	73.59	65.37	43.58
μ	75.80	65.83	42.82
σ	2.20	3.13	1.86

Table A.8: Comparison table for dataset Pima.

Sonar

Seed	SVM approach	A1B0 approach	SVM sparsity
1	65.08	85.71	0.69
2	63.49	77.78	0.69
3	60.32	82.54	1.38
4	58.73	82.54	0.69
5	60.32	90.48	0.00
6	63.49	80.95	1.38
7	65.08	88.89	0.00
8	49.21	71.43	0.69
9	65.08	88.89	0.69
10	63.49	84.13	2.07
11	57.14	85.71	0.69
12	63.49	82.54	0.00
13	47.62	84.13	0.69
14	68.25	88.89	0.69
15	52.38	80.95	1.38
16	42.86	80.95	0.69
17	50.79	82.54	0.00
18	60.32	90.48	0.00
19	61.90	85.71	0.00
20	61.90	84.13	0.69
21	57.14	76.19	0.69
22	66.67	90.48	0.69
23	69.84	88.89	0.69
24	50.79	77.78	0.69
25	46.03	77.78	0.00
26	66.67	87.30	0.69
27	63.49	77.78	0.69
28	53.97	74.60	1.38
29	50.79	79.37	0.69
30	65.08	90.48	0.69
31	57.14	90.48	1.38
32	58.73	87.30	1.38
33	74.60	88.89	0.69
34	55.56	85.71	0.00
35	53.97	80.95	0.69
36	63.49	82.54	0.69
37	55.56	77.78	0.69
38	63.49	85.71	0.00
39	66.67	90.48	1.38
40	50.79	80.95	0.00
41	58.73	71.43	0.00
42	60.32	85.71	0.00
43	68.25	87.30	0.00
44	74.60	80.95	0.00
45	66.67	87.30	0.00
46	55.56	82.54	0.69
47	57.14	84.13	0.69
48	57.14	73.02	0.69
49	58.73	84.13	0.69
50	50.79	87.30	0.69
μ	59.59	83.49	0.61
σ	7.06	5.15	0.50

Table A.9: Comparison table for dataset Sonar.

Wdbc Mangasarian and Wolberg

Seed	**SVM approach**	**A1B0 approach**	**SVM sparsity**
1	97.08	91.23	56.17
2	97.66	96.49	53.40
3	94.15	92.98	55.67
4	97.66	97.08	54.41
5	95.91	94.15	54.66
6	98.25	94.74	55.16
7	95.32	92.98	56.42
8	95.32	95.91	53.90
9	94.74	91.81	52.90
10	97.08	95.32	53.40
11	97.66	90.64	53.15
12	95.91	90.06	55.16
13	98.25	97.08	55.42
14	94.15	96.49	55.92
15	94.15	93.57	54.91
16	97.66	89.47	55.16
17	94.74	93.57	54.16
18	97.08	92.98	55.67
19	95.91	93.57	54.66
20	94.15	94.15	56.68
21	91.23	88.30	58.19
22	95.32	92.98	55.16
23	97.66	88.89	55.16
24	94.74	92.40	53.90
25	90.06	91.23	56.42
26	95.32	91.81	54.91
27	96.49	91.81	54.91
28	95.91	90.64	55.42
29	92.40	90.06	54.41
30	95.32	94.74	54.41
31	95.32	92.40	54.66
32	96.49	90.06	54.91
33	94.74	96.49	53.65
34	93.57	94.15	54.41
35	95.91	89.47	55.92
36	97.08	93.57	55.16
37	95.91	89.47	56.17
38	92.40	91.81	56.93
39	96.49	90.64	54.16
40	96.49	91.23	55.16
41	98.25	94.74	55.16
42	94.15	87.72	56.42
43	92.40	91.81	55.67
44	92.98	92.40	55.42
45	95.32	87.13	56.42
46	97.66	95.32	54.41
47	95.91	94.74	54.16
48	95.32	94.74	55.16
49	96.49	94.74	55.16
50	95.91	94.15	55.42
μ	95.49	92.68	55.09
σ	1.88	2.51	1.04

Table A.10: Comparison table for dataset Wdbc Mangasarian and Wolberg.

Wpbc Mangasarian and Wolberg

Seed	SVM approach	A1B0 approach	SVM sparsity
1	74.58	74.58	9.63
2	77.97	79.66	9.63
3	77.97	79.66	8.89
4	79.66	77.97	9.63
5	69.49	69.49	8.89
6	77.97	81.36	9.63
7	77.97	79.66	5.93
8	77.97	79.66	11.11
9	72.88	72.88	11.11
10	81.36	83.05	8.89
11	76.27	76.27	12.59
12	69.49	69.49	15.56
13	72.88	72.88	12.59
14	83.05	83.05	3.70
15	76.27	79.66	8.89
16	79.66	84.75	8.15
17	69.49	72.88	13.33
18	67.80	69.49	11.11
19	77.97	77.97	11.85
20	71.19	74.58	14.07
21	79.66	77.97	10.37
22	71.19	72.88	11.11
23	67.80	67.80	7.41
24	67.80	67.80	7.41
25	71.19	69.49	16.30
26	77.97	74.58	8.15
27	79.66	77.97	6.67
28	74.58	76.27	7.41
29	67.80	69.49	8.15
30	71.19	69.49	10.37
31	71.19	72.88	10.37
32	69.49	72.88	11.85
33	67.80	69.49	11.11
34	83.05	83.05	9.63
35	84.75	84.75	8.15
36	81.36	81.36	8.15
37	88.14	86.44	5.19
38	72.88	72.88	8.15
39	77.97	79.66	9.63
40	74.58	74.58	6.67
41	74.58	76.27	7.41
42	77.97	81.36	7.41
43	77.97	77.97	10.37
44	76.27	76.27	11.11
45	72.88	74.58	10.37
46	69.49	71.19	11.85
47	72.88	74.58	10.37
48	66.10	66.10	9.63
49	77.97	77.97	8.15
50	76.27	79.66	8.15
μ	75.09	75.93	9.65
σ	5.06	5.04	2.44

Table A.11: Comparison table for dataset Wpbc Mangasarian and Wolberg.

Appendix B

A statistical tool: Wilcoxon Test

Wilcoxon Test is a nonparametric test used to compare two different series of measures obtained on the same items. With that test we can compare two different instruments or analytical procedure or operators.

There are tree different types of what is known in the scientific literature as *Wilcoxon test*:

- Wilcoxon Two Sample Test;
- Wilcoxon Matched-Pairs Test;
- Wilcoxon Signed Rank Sum Test.

The test we will describe, that is the one we used, is the last one. Wilcoxon Paired Signed Rank test is a nonparametric evaluation of paired differences. Pairs of measurements forms the raw data, and the difference between the two members of the pair is used to calculate the statistics. Wilcoxon is therefore the nonparametric equivalent of the paired t test.

B.1 Prerequisites and hypothesis

Input data distribution. The test can be used on data that have a Gaussian distribution or on data that represent ordinal variables (recalling, from the theory, that an *ordinal variable* is a variable whose value set V is a linearly ordered set);

Measures on the same data. It is necessary that the items, on which the methods we are comparing are used, form exactly the same instances set for the application of both the measurements;

Null hypothesis. We state that:

H_0 : the first instances set is equal to the second one.

That is: there is not a statistically significative difference between the use of the first or the second method on the items.

Number of measurements. It is necessary to have more than a certain number of measurements: usually between a few dozen and the Student threshold.

B.2 Procedure

To perform the test, the following steps have to be done:

- The test begins with the transformation of each couple of values ($measure_1 = X_{1i}$, $measure_2 = X_{2i}$) into the absolute value of the difference between them:

$$|X_{1i} - X_{2i}| \quad \forall i$$

- Couples that produces null differences do not give any kind of information in the test, so they can be eliminated

$$if |X_{1i} - X_{2i}| \quad then\ entry_i\ isremoved$$

- The obtained differences are ordered for increasing absolute values. And then a rank (unsigned quantity) is assigned to each of them, through a special procedure

$$|X_{1i} - X_{2i}| \quad : \quad R_i$$

- A sign is reassigned to each rank, depending on whether $X_{1i} - X_{2i}$ was originally positive or negative

$$if X_{1i} - X_{2i} > 0 \ \ then\ R_i^{'} = R_i$$

$$if X_{1i} - X_{2i} < 0 \ \ then\ R_i^{'} = -R_i$$

- The Wilcoxon test statistic value W is computed as the sum of the signed ranks

$$W = \sum_i R_i^{'}$$

It can be demonstrated that, for samples with $n_{diff} > 20$, the test statistic W is approximately normally distributed with mean μ_W and standard deviation σ_W.

The mean value of W is given by

$$\mu_W = \frac{n_{diff}(n_{diff} + 1)}{4}$$

and its standard deviation by

$$\sigma_W = \sqrt{\frac{n_{diff}(n_{diff} + 1)(2n_{diff} + 1)}{24}}$$

It is now possible to normalized the W distribution

$$Z = \frac{W - \frac{n_{diff}(n_{diff}+1)}{4}}{\sqrt{\frac{n_{diff}(n_{diff}+1)(2n_{diff}+1)}{24}}}$$

Then, knowing the value of a certain occurrence of W, the relative value of Z can be computed.

Recalling the concept of *Rejection Region*:

> The rejection region is used in hypothesis testing. Let T be a test statistic. Possible values of T can be divided into two regions: the *acceptance region* and the *rejection region.* If the value of T comes out to be in the acceptance region, the null hypothesis (the one being tested) is accepted, or at any rate not rejected. If T falls in the rejection region, the null hypothesis is rejected.

Thus, null hypothesis is rejected if the computed Z falls in the rejection region. E.g., referring to the standard Gaussian distribution[1] and assuming to accept a statement with a statistical significativeness greater of equal to 95%, the decision rule states that:

H_0 is rejected if $Z > 1.96$ or $Z < -1.96$
otherwise H_0 is accepted.

B.3 An example to describe the mathematical steps of the test

To make more clear the whole procedure, we extracted a small sample of measurements we actually compared in our work. We have 20 couples of measurements of accuracy calculated on as many dataset instances of Wpbc Mangasarian and Wolberg, obtained randomly.

In table B.1 we have:

- **Id** or seed, it represents the identifier of the specific instance randomly defined on the dataset;
- $\mathbf{X_1}$, it represents the first accuracy measurement, the one made with classic SVM approach;
- $\mathbf{X_2}$, it represents the second accuracy measurement, the one made using just the simple classifier, without solving mathematical models;

[1]In Appendix F the normal standard distribution table.

id	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_1 - X_2}$	$\mathbf{\lvert X_1 - X_2\rvert}$	**rank**	**signed rank**
1	74.58	74.58	0	0	-	-
2	79.66	77.97	1.69	1.69	1/7	1/7
3	79.66	77.97	1.69	1.69	1/7	1/7
4	77.97	79.66	-1.69	1.69	1/7	-1/7
5	69.49	69.49	0	0	-	-
6	81.36	77.97	3.39	3.39	1.25	1.25
7	79.66	77.97	1.69	1.69	1/7	1/7
8	79.66	77.97	1.69	1.69	1/7	1/7
9	72.88	72.88	0	0	-	-
10	81.36	83.05	-1.69	1.69	1/7	-1/7
11	76.27	76.27	0	0	-	-
12	69.49	69.49	0	0	-	-
13	72.88	72.88	0	0	-	-
14	83.05	83.05	0	0	-	-
15	79.66	76.27	3.39	3.39	1.25	1.25
16	79.66	84.75	-5.09	5.09	3	-3
17	72.88	69.49	3.39	3.39	1.25	1.25
18	67.80	69.49	-1.69	1.69	1/7	-1/7
19	77.97	77.97	0	0	-	-
20	71.19	74.58	-3.39	3.39	1.25	-1.25

Table B.1: Wilcoxon test table for a small sample on Wpbc Mangasarian and Wolberg dataset.

- $\mathbf{X_1} - \mathbf{X_2}$, it represents the signed difference between the two measurements;
- $|\mathbf{X_1} - \mathbf{X_2}|$, it represents the absolute value of the difference between the two measurements;
- **rank**, it represents the rank assigned to the unsigned difference (the not null unsigned differences are first of all increasing ordered: if there are k different possible values for the differences, then the ranks will belong to the interval $[1, k]$; if the jth possible value for the unsigned difference, in the previously described order, belongs to k_j different couple than the rank for them is given by $(j-1) + \frac{1}{k_j}$ – e.g. in the table there 3 possible values for the unsigned differences, so the ranks will belong to $[1, 3]$, since $3,39$ is the second value, in increasing order, and it belongs to 4 couples, the associated rank value is $(2-1) + \frac{1}{4} = 1,25$);
- **signed rank**, it represents the rank with the original difference sign.

In the following lets perform Wilcoxon test on the data in B.1, step by step. Since that $n_{diff} = 11$, in this case, we can compute

$$\mu_W = 33 \quad and \quad \sigma_W = \sqrt{126,5}$$

And also

$$W = \sum_i rank_i = \frac{1}{14}$$

Then

$$Z = \frac{\frac{1}{14} - 33}{\sqrt{126,5}} \simeq -2,93$$

From the normal standard distribution table we obtain

$$p = p[Z < -2,93 \,||\, Z > 2,93] = 1 - p[|Z| > 2,93] = 1 - 2 \cdot Z^{-1}(2,93) \simeq 0,0034 \ll 0,05$$

that tells us that the null hypothesis H_0 is rejected with a statistical confidence of 99,66%: the two compared methods are statistically significatively different.

The last thing to do now is to understand which of the two methods wins and which loses.

The simplest approach to realize that is to calculate the sign of the mean values of the signed differences.

Appendix C

More detailed results

In the following we report some detailed results obtained during our several experimental tests.

In table C.1 we can find the detailed accuracies obtained for each couple of parameters (C, γ), with the classic *SVM approach*, equipped with Gaussian kernel. The values have been obtained without performing any kind of cross validation, just solving the classic quadratic model for the training procedure on the training set and classifying, with the computed parameters, the test set items. As previously noticed, the value of parameter γ that generally gives the best accuracies on our normalized is $\gamma = 0.1$.

In table C.2 we can find the accuracies obtained at the variation of parameter C, with the classic *SVM approach*, equipped with linear kernel. The values have been obtained without performing any kind of cross validation, just solving the classic quadratic model for the training procedure on the training set and classifying, with the computed parameters, the test set items.

In table C.3 we report the detailed results obtained, for each dataset, for the first evaluation of the use of a MIP model in *SVM approach* with linear kernel. The description and the purposes of these last tests are given in section 4.2.2.

Australian					
C \ γ	**0.1**	**1**	**10**	**100**	**1000**
0.01	54.59	54.59	54.59	54.59	54.59
0.1	54.59	54.59	54.59	54.59	54.59
1	84.06	57.00	54.11	54.59	54.59
10	81.64	56.52	54.59	54.11	54.59
100	76.81	57.00	54.59	54.11	54.59

Breast Mangasarian and Wolberg

C \ γ	0.1	1	10	100	1000
0.01	95.61	62.44	62.44	62.44	62.44
0.1	96.10	93.17	62.44	62.44	62.44
1	97.07	95.61	84.39	71.71	71.71
10	95.61	97.07	86.83	71.71	71.71
100	94.63	97.07	86.83	71.71	71.71

Bupa

C \ γ	0.1	1	10	100	1000
0.01	52.88	52.88	52.88	52.88	52.88
0.1	52.88	52.88	52.88	52.88	52.88
1	64.42	65.38	53.85	53.85	53.85
10	64.42	65.38	56.73	53.85	53.85
100	62.50	62.54	56.73	53.85	53.85

Heart

C \ γ	0.1	1	10	100	1000
0.01	62.96	62.96	62.96	62.96	62.96
0.1	76.54	62.96	62.96	62.96	62.96
1	79.01	64.20	62.96	62.96	62.96
10	75.31	64.20	62.96	62.96	62.96
100	75.31	64.20	62.96	62.96	62.96

Ionosphere

C \ γ	0.1	1	10	100	1000
0.01	63.21	63.21	63.21	63.21	63.21
0.1	65.09	63.21	63.21	63.21	63.21
1	93.40	66.04	63.21	63.21	63.21
10	92.45	66.04	63.21	63.21	63.21
100	90.57	66.04	63.21	63.21	63.21

Pima

γ **C**	**0.1**	**1**	**10**	**100**	**1000**
0.01	68.40	68.40	68.40	68.40	68.40
0.1	74.46	68.40	68.40	68.40	68.40
1	73.16	71.00	68.40	68.40	68.40
10	74.46	67.53	68.40	68.40	68.40
100	66.67	67.53	68.40	68.40	68.40

Sonar

γ **C**	**0.1**	**1**	**10**	**100**	**1000**
0.01	57.14	57.14	57.14	57.14	57.14
0.1	57.14	57.14	57.14	57.14	57.14
1	80.95	57.14	57.14	57.14	57.14
10	80.95	57.14	57.14	57.14	57.14
100	80.95	57.14	57.14	57.14	57.14

Wdbc Mangasarian and Wolberg

γ **C**	**0.1**	**1**	**10**	**100**	**1000**
0.01	61.40	61.40	61.40	61.40	61.40
0.1	95.91	61.40	61.40	61.40	61.40
1	98.25	61.99	61.40	61.40	61.40
10	97.08	61.99	61.40	61.40	61.40
100	96.49	61.99	61.40	61.40	61.40

Wpbc Mangasarian and Wolberg

γ **C**	**0.1**	**1**	**10**	**100**	**1000**
0.01	77.97	77.97	77.97	77.97	77.97
0.1	77.97	77.97	77.97	77.97	77.97
1	79.66	77.97	77.97	77.97	77.97
10	77.97	77.97	77.97	77.97	77.97
100	77.97	77.97	77.97	77.97	77.97

Table C.1: Accuracies for *SVM approach*, Gaussian kernel.

Australian					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	82.61	80.68	79.23	80.19	79.19

Breast Mangasarian and Wolberg					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	97.07	96.10	96.59	96.59	96.10

Bupa					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	53.85	59.62	66.35	66.35	66.35

Heart					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	80.25	80.25	81.48	79.01	81.48

Ionosphere					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	90.57	91.51	91.51	89.62	89.62

Pima					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	74.46	71.43	71.00	71.00	71.00

Sonar					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	84.13	80.95	76.19	76.19	76.19

Wdbc Mangasarian and Wolberg					
C	**0.01**	**0.1**	**1**	**10**	**100**
Accuracy	95.32	97.08	97.08	95.91	95.91

Wpbc Mangasarian and Wolberg					
C	**0.01**	**0.1**	**1**	**10**	**100**

Accuracy	74.58	76.27	74.58	77.97	74.58

Table C.2: Accuracies for *SVM approach*, linear kernel.

Australian

S C	**1**	**10**	**100**	**1000**
0.01	83.09	83.57	82.61	82.61
0.1	83.09	83.09	81.64	81.16
1	83.09	83.09	82.13	82.13
10	83.57	82.13	81.16	81.16
100	81.16	81.64	81.16	81.16

Breast Mangasarian and Wolberg

S C	**1**	**10**	**100**	**1000**
0.01	96.59	97.07	97.07	97.07
0.1	96.59	96.59	93.59	97.07
1	96.59	96.10	96.10	96.10
10	96.59	96.10	96.10	96.10
100	96.59	96.10	96.10	96.10

Bupa

S C	**1**	**10**	**100**	**1000**
0.01	53.85	64.42	60.58	61.54
0.1	56.73	61.54	61.54	61.54
1	59.62	65.38	63.46	64.46
10	59.62	64.42	63.46	63.46
100	59.62	64.42	63.46	63.46

Heart

S C	**1**	**10**	**100**	**1000**
0.01	76.54	81.48	87.65	87.65
0.1	90.12	79.01	79.01	79.01
1	77.78	83.95	92.72	81.48
10	77.78	81.48	81.48	82.72

100	77.78	81.48	81.48	81.48

Ionosphere

C \ S	**1**	**10**	**100**	**1000**
0.01	78.30	88.86	88.68	88.68
0.1	89.62	90.57	92.45	92.45
1	89.62	91.51	91.51	91.51
10	92.45	90.57	91.51	91.51
100	89.62	91.51	89.62	89.62

Pima

C \ S	**1**	**10**	**100**	**1000**
0.01	70.13	71.43	70.56	69.70
0.1	70.13	71.00	71.00	71.00
1	72.73	71.86	71.00	71.00
10	72.73	71.86	71.00	71.00
100	72.73	71.86	71.00	71.00

Sonar

C \ S	**1**	**10**	**100**	**1000**
0.01	52.38	82.54	82.54	82.54
0.1	74.60	80.95	82.54	82.54
1	77.78	74.60	76.19	76.19
10	79.37	74.60	74.60	74.60
100	77.78	76.19	74.60	74.60

Wdbc Mangasarian and Wolberg

C \ S	**1**	**10**	**100**	**1000**
0.01	94.15	96.49	97.08	97.08
0.1	97.08	97.66	97.08	97.08
1	96.49	97.08	97.08	97.08
10	97.66	96.49	96.49	96.49
100	95.32	95.91	95.91	95.91

Wpbc Mangasarian and Wolberg				
C \ **S**	**1**	**10**	**100**	**1000**
0.01	74.58	74.58	61.02	59.32
0.1	74.58	61.02	61.02	61.02
1	72.88	55.93	54.24	54.24
10	72.88	66.10	57.63	55.93
100	74.58	67.80	66.80	66.10

Table C.3: Detailed accuracies for MIP alternative approach with linear kernel.

Appendix D

Scripts

D.1 Example 1

In this section we report an example of a script for the simulation of the *SVM approach* on dataset Heart.

Script D.1: Script for *SVM approach* with 5-fold cross validation.

```
#!/bin/bash
cd ..
cd ..
cd CODICI/
gamma=(0.1 1 10 100 1000)
C=(0.01 0.1 1 10 100)
filedimensione=''n_feature.txt''
n_item=270
for (( r=1; r<=3; r++ ))
    do
        file3accuracy=''accuracy_finale_''$r$''.txt''
        gcc Parser1_HEART.c -o Parser1_HEART -lm
        ./Parser1_HEART $r
        filetrainingFIN=''File_txt/PerSVMlight/input_TrS_SVM_HEART_norm.txt''
        filetestFIN=''File_txt/PerSVMlight/input_TeS_SVM_HEART_norm.txt''
        for (( l=1; l<=5; l++ ))
                do
                for (( j=0; j <=${#gamma[*]}-1; j++ ))
                        do
                        for (( k=0; k <=${#C[*]}-1; k++ ))
                                do
                                fileaccuracy=''File_txt/Accuracy/accuracy''${gamma[j]}$''_''${C[k]}$''_''$l$''.txt''
                                filetraining=''File_txt/PerSVMlight/tutti_meno_''$l$''.txt''
                                filetest=''File_txt/PerSVMlight/''$l$''.txt''
                                filenumelementi=''File_txt/PerSVMlight/num_elementi''$l$''.txt''
                                echo ''esecuzione gamma'' ''${gamma[j]}, '' '' C'' ''${C[k]}''
                                gcc ModelConstructor_SVM.c -o ModelConstructor_SVM -lm
                ./ModelConstructor_SVM ${gamma[j]} ${C[k]} $filetraining $filenumelementi $filedimensione
                cplex < inputcpx.dat
                gcc Parser2.c -o Parser2 -lm
                ./Parser2 $filenumelementi $filedimensione $filetraining $filetest ${gamma[j]} $fileaccuracy
                cd File_txt/
                DAESCLUDERE=dataset_HEART.txt

                for i in $( ls );
                    do
                    if [[ $i == *.txt ]]
                        then
                        if [[ $i != $DAESCLUDERE ]]
                            then
                            rm $i
                        fi
                    fi
                    if [[ $i == *.lp ]]
                        then
                        rm $i
                    fi
                    if [[ $i == *.sol ]]
                        then
                        rm $i
                    fi
                    done

                cd ..
                rm ModelConstructor_SVM
                rm Parser2

                done
            done
        done
    gcc MaxCouple.c -o MaxCouple -lm
    ./MaxCouple
    gcc ModelConstructorFIN_SVM.c -o ModelConstructorFIN_SVM -lm
    ./ModelConstructorFIN_SVM $filetrainingFIN $filedimensione $n_item
    cplex < inputcpx.dat
    gcc Parser2FIN.c -o Parser2FIN -lm
    ./Parser2FIN  $filedimensione $filetrainingFIN $filetestFIN $file3accuracy $n_item

    cd File_txt/
    DAESCLUDERE=dataset_HEART.txt
    for i in $( ls );
            do
            if [[ $i == *.txt ]]
                then
                    if [[ $i != $DAESCLUDERE ]]
                        then
                            rm $i
                    fi
            fi
            if [[ $i == *.lp ]]
                then
                    rm $i
            fi
            if [[ $i == *.sol ]]
                then
                    rm $i
            fi
        done
    cd PerSVMlight/
    for i in $( ls );
        do
        if [[ $i == *.txt ]]
            then
            rm $i
        fi
        done
    cd ..
    cd Accuracy/
    for i in $( ls );
        do
        if [[ $i == *.txt ]]
            then
            rm $i
        fi
        done
    cd ..
    cd ..

    rm Parser1_HEART
    rm MaxCouple
    rm ModelConstructorFIN_SVM
    rm Parser2FIN
    done

gcc MeanFinalAccuracy.c -o MeanFinalAccuracy -lm
./MeanFinalAccuracy

rm accuracy_finale_1.txt
rm accuracy_finale_2.txt
rm accuracy_finale_3.txt
rm MeanFinalAccuracy
rm param_max_acc.txt
rm n_feature.txt
rm *.log

cd ..
cd X_SCRIPT
```

We can see that the execution of the code involves 4 for cycles:

-
  ```
      for (( r=1; r <=3; r++ ))
  ```

  enables us to repeat the whole procedure of classification, including the 5-fold cross validation, 3 times, to compute the final mean accuracy;

- `for (( l=1; l <=5; l++ ))`

  enables us to realize 5-fold cross validation (lines 22-25 rename properly at each iteration training and validation sets in order to change them correctly);

- `for (( j=0; j <=${#gamma [*]} -1; j++ ))`

  enables us to try all the values of γ, defined at line 5;

- `for (( k=0; k <=${#C[*]} -1; k++ ))`

  enables us to try all the values of C, defined at line 6.

Scripts for all the other mathematical models used in out tests in general differs from this one for the model constructor called at lines 27-28 and 63-64.

At lines 29 and 65 we can see a call of the optimization solver CPLEX, with a redirection of the input from the file `inputcpx.dat`.

An example of that file can be as the one in listing D.2 (actually, since it contains MIP settings, this `.dat` refers to the resolution of a MIP model).

Listing D.2: File with CPLEX settings.

```
set mip int tol 0
set timelimit 1200
set mip pol time 900
r ModelSVM_HEART.lp
opt
w Sol_ModelSVM_HEART.sol
disp pro var a* > Alpha\a.txt
quit
```

It contains some settings used for the optimizer and the instructions for read, optimize and write in a `.sol` file the got solution.

Lines 35-52 and 71-105 permit to remove all the files produced during the computation, including occupying space `.log` files or `.txt` files written for debugging but also, and it is a fundamental action, to remove the `.sol` file just finished to use. This is important because CPLEX asks the user if he/she wants to overwrite a solution file, and the software considers "no" to be the default answer. Since in our iterations we re-use the same `input.dat` file, we need to remove the previous .sol file in the working directory.

D.2 Example 2

In this section an example of a script for the fifty executions of the *SVM approach* in parallel with *A1B0 approach* on fifty different instances of dataset HEART, without the validation procedure.

Script D.3: Script for the determination of the accuraries by *SVM approach* and *A1B0 approach*.

```
#!/bin/bash
cd ..
cd ..
cd CODICI
n_item=270
n_feature=''n_feature.txt''
for (( r=1; r<=50; r++ ))
      do
      echo ''seed'' $r
      gcc Parser1_HEART.c -o Parser1_HEART -lm
      ./Parser1_HEART $r
      filetrainingset=''File_txt/PerSVMlight/input_TrS_SVM_HEART_norm.txt
          ''
      filetestset=''File_txt/PerSVMlight/input_TeS_SVM_HEART_norm.txt''
      gcc Parser2_A1B0.c -o Parser2_A1B0 -lm
      ./Parser2_A1B0 $n_item $n_feature $filetrainingset $filetestset
      done

cd File_txt
for i in $( ls );
      do
      if [[ $i == *.txt ]]
            then
            rm $i
      fi
      done
cd PerSVMlight
for i in $( ls );
      do
      if [[ $i == *.txt ]]
            then
            rm $i
      fi
      done
cd ..
cd ..
cd ..
cd X_SCRIPT
```

Appendix E

C Code

In the following we report a listing of C code. It enables us to format the `.lp` file that contains the model to be sent in input to CPLEX optimizer.

E.1 Example of `Model constructor`

Code E.1: Code for formatting .lp model.

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[]){

        int i, j, elemento1;
        double elemento;
        char car;

    //number of dataset elements
        int n_elementi = atoi(argv[3]);
        int n_train = (n_elementi*7)/10;

    //number of attributes (after preprocessing)
        int n_dimensione;
        FILE *file_dim = fopen(argv[2], ''r'');
        fscanf(file_dim, ''%d'', &elemento1);
        n_dimensione = elemento1;
        fclose(file_dim);

    //optimal couple (\gamma, C) from validation procedure
        FILE *file_param = fopen(''param_max_acc.txt'', ''r'');
        fscanf(file_param, ''%lf'', &elemento);
        double gamma = elemento;
        fscanf(file_param, ''%lf'', &elemento);
        double C = elemento;
        fclose(file_param);

    //training set
        FILE *file_trainingset;
        file_trainingset = fopen(argv[1], ''r'');

    /* reading and saving training set items */

        double **x;
        x = (double**)malloc(n_train * sizeof(double) );
        for(i=0; i<n_train; i++)
                x[i] = (double*)malloc(n_dimensione * sizeof(double) );

        int *y;
        y = (int*)malloc(n_train * sizeof(int));

        for (i = 0; i < n_train; i++){
                fscanf(file_trainingset, ''%d'', &elemento1);
                y[i] = elemento1;
                for (j = 0; j < n_dimensione; j++){
                        fscanf(file_trainingset, ''%d'', &elemento1);
                        fscanf(file_trainingset, ''%c'', &car);
                        fscanf(file_trainingset, ''%lf'', &elemento);
                        x[i][j] = elemento;
                }
        }

        fclose(file_trainingset);

        /* computing kernel matrix */

        double **kernel;
        kernel = (double**)malloc(n_train * sizeof(double) );
        for(i=0; i<n_train; i++)
                kernel[i] = (double*)malloc(n_train * sizeof(double) );

        int indice, l;

        for(i = 0; i < n_train; i++){
                for (j = 0; j < n_train; j++){
                        kernel[i][j] = 0;
                }
        }

        for (indice=0; indice < n_train; indice++){
                for(j = 0; j < n_train; j++){
                        double somma = 0;;
                        for(i = 0; i < n_dimensione; i++){
                                somma = somma + (x[indice][i] - x[j][i])*(
                                    x[indice][i] - x[j][i]);
                        }
                        kernel[indice][j] = - somma*gamma;
                        kernel[indice][j] = exp(kernel[indice][j]);
                }
        }

    /* computing model coefficients */

        double **coeff;
        coeff = (double**)malloc(n_train * sizeof(double) );
        for(i=0; i<n_train; i++)
                coeff[i] = (double*)malloc(n_train * sizeof(double) );

        for (i = 0; i < n_train; i++){
                for (j = 0; j < n_train; j++){
                        coeff[i][j] = (double) y[j]*y[i]*kernel[i][j];
                }
        }

        /* constructing SVM model in .lp format */

        FILE *file_lp;
        file_lp = fopen(''File_txt/Model_SVM.lp'', ''w'');

                /* objective function */

        fprintf(file_lp, ''MINIMIZE\n'');
        fprintf(file_lp, '' OBJ: '');
        j = 0;
        fprintf(file_lp, ''%.10lf x%d '', C, j+1);
        for (j=1; j<n_train; j++){
                fprintf(file_lp, ''+ %.10lf x%d '', C, j+1);
        }

        for(i = 0; i < n_train; i++){
                for (j = 0; j < n_train; j++){
                        if (i == j){     // \lambda^2 terms
                                if (i == 0){     //if first open [
                                        if (coeff[i][j] > 0){
                                                fprintf(file_lp, ''+ [%.20
                                                    lf a%d^2 '', coeff[i][j
                                                    ], i+1);
                                        }
                                        else{
                                                double cff = - coeff[i][j
                                                    ];
                                                fprintf(file_lp, ''+
                                                    [-%.20lf a%d^2 '', cff,
                                                     i+1);
                                        }
                                }
                                else if (i == n_train-1){      //if last
                                    close ]
                                        if (coeff[i][j] >
                                            0.000000000000001){
                                                fprintf(file_lp, ''+ %.20
                                                    lf a%d ^ 2] / 2\n'',
                                                    coeff[i][j], i+1);
                                        }
                                        else if (coeff[i][j] <
                                            -0.000000000000001){
                                                double cff = - coeff[i][j
                                                    ];
                                                fprintf(file_lp, ''- %.20
                                                    lf a%d ^ 2] / 2\n'',
                                                    cff, i+1);
                                        }
                                }
                                else{
```

```
                    if (coeff[i][j] > 0.000000000000001){
                        fprintf(file_lp, ''+ %.20lf a%d ^ 2 '', coeff[i][j], i+1);
                    }
                    else if (coeff[i][j] < -0.000000000000001){
                        double cff = - coeff[i][j];
                        fprintf(file_lp, ''- %.20lf a%d ^ 2 '', cff, i+1);
                    }
                }
            }
            else{       // \lambda_i*\lambda_j i<>j terms
                if (coeff[i][j] > 0.000000000000001){
                    fprintf(file_lp, ''+ %.20lf a%d * a%d '', coeff[i][j], i+1, j+1);
                }
                else if (coeff[i][j] < -0.000000000000001){
                    double cff = - coeff[i][j];
                    fprintf(file_lp, ''- %.20lf a%d * a%d '', cff, i+1, j+1);
                }
            }
        }
    }

  /* bounds */

    fprintf(file_lp, ''SUBJECT TO\n'');
    for(i = 0; i < n_train; i++){
        int k = i+1;
        fprintf(file_lp, '' V%d: '', k);
            for (j = 0; j < n_train; j++){
            int h = j+1;
            if (coeff[i][j] > 0.000000000000001){    //non negative coefficient
                if (j == 0){    //if first without sign
                    fprintf(file_lp, ''%.20lf a%d '', coeff[i][j], h);
                }
                else{   //if not with sign
                    fprintf(file_lp, ''+ %.20lf a%d '', coeff[i][j], h);
                }
            }
            else if (coeff[i][j] < -0.000000000000001){  // negative coefficient
                if (j == 0){
                    fprintf(file_lp, ''%.20lf a%d '', coeff[i][j], h);
                }
                else{
```

```
                                        double cff = coeff[i][j]*(-1);
                                        fprintf(file_lp, ''- %.20lf a%d ''
                                            , cff, h);
                                }
                        }
                }
                if (y[i] > 0)
                        fprintf(file_lp, ''+ b + x%d >= 1\n'', i+1);
                else
                        fprintf(file_lp, ''- b + x%d >= 1\n'', i+1);
        }

          /* variables */

        fprintf(file_lp, ''BOUNDS\n'');
        for(i = 0; i < n_train; i++){
                int k = i+1;
                fprintf(file_lp, '' a%d >= 0\n'', k);
        }
        fprintf(file_lp, '' - infinity <= b <= + infinity \n'');

        for(i = 0; i < n_train; i++){
                int k = i+1;
                fprintf(file_lp, '' x%d >= 0\n'', k);
        }

    /* closure tag */

        fprintf(file_lp, ''END\n'');

        fclose(file_lp);

        return 0;
}
```

Appendix F

Table of the normal distribution

z	**0.00**	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**
0.0	.00000	.00399	.00792	.01197	.01595	.01994	.02392	.02790	.03188	.03586
0.1	.03983	.04380	.04776	.05172	.05567	.05962	.06356	.06749	.07142	.07535
0.2	.07926	.08317	.08706	.09095	.09483	.09871	.10257	.10642	.11026	.11409
0.3	.11791	.12172	.12552	.12930	.13307	.13683	.14058	.14431	.14803	.15173
0.4	.15542	.15910	.16276	.16640	.17003	.17364	.17724	.18082	.18439	.18793
0.5	.19146	.19497	.19847	.20194	.20540	.20884	.21226	.21566	.21904	.22240
0.6	.22575	.22907	.23237	.23565	.23891	.24215	.24537	.24857	.25175	.25490
0.7	.25804	.26115	.26424	.26730	.27035	.27337	.27637	.27935	.28230	.28524
0.8	.28814	.29103	.29389	.29673	.29955	.30234	.30511	.30785	.31057	.31327
0.9	.31594	.31859	.32121	.32381	.32639	.32794	.33147	.33398	.33646	.33891
1.0	.34134	.34375	.34614	.34849	.35083	.35314	.35543	.35769	.35993	.36214
1.1	.36433	.36650	.36864	.37076	.37286	.37493	.37698	.37900	.38100	.38298
1.2	.38493	.38686	.38877	.39065	.39251	.39435	.39617	.39796	.39973	.40147
1.3	.40320	.40490	.40658	.40824	.40988	.41149	.41309	.41466	.41621	.41774
1.4	.41924	.42073	.42220	.42364	.42507	.42647	.42786	.42922	.43056	.43189
1.5	.43319	.43448	.43574	.43699	.43822	.43943	.44062	.44179	.44295	.44408
1.6	.44520	.44630	.44738	.44845	.44950	.45053	.45154	.45254	.45352	.45449
1.7	.45543	.45637	.45728	.45818	.45907	.45994	.46080	.46164	.46246	.46327
1.8	.46407	.46485	.46562	.46637	.46712	.46784	.46856	.46926	.46995	.47062
1.9	.47128	.47193	.47257	.47320	.47381	.47441	.47500	.47558	.47615	.47670
2.0	.47725	.47778	.47831	.47882	.47932	.47982	.48030	.48077	.48124	.48169
2.1	.48214	.48257	.48300	.48341	.48382	.48422	.48461	.48500	.48537	.48574
2.2	.48610	.48645	.48679	.48713	.48745	.48778	.48809	.48840	.48870	.48899
2.3	.48928	.48956	.48983	.49010	.49036	.49061	.49086	.49111	.49134	.49158
2.4	.49180	.49202	.49224	.49245	.49266	.49286	.49305	.49324	.49343	.49361
2.5	.49379	.49396	.49413	.49430	.49446	.49461	.49477	.49492	.49506	.49520
2.6	.49534	.49547	.49560	.49573	.49585	.49598	.49609	.49621	.49632	.49643
2.7	.49653	.49664	.49674	.49683	.49693	.49702	.49711	.49720	.49728	.49736
2.8	.49745	.49752	.49760	.49767	.49774	.49781	.49788	.49795	.49801	.49807
2.9	.49813	.49819	.49825	.49831	.49836	.49841	.49846	.49851	.49856	.49861
3.0	.49865	.49869	.49874	.49878	.49882	.49886	.49889	.49893	.49897	.49900
3.1	.49903	.49906	.49910	.49913	.49916	.49918	.49921	.49924	.49926	.49929
3.2	.49931	.49934	.49936	.49938	.49940	.49942	.49944	.49946	.49948	.49950
3.3	.49952	.49953	.49955	.49957	.49958	.49960	.49961	.49962	.49964	.49965
3.4	.49966	.49968	.49969	.49970	.49971	.49972	.49973	.49974	.49975	.49976
3.5	.49977	.49978	.49978	.49979	.49980	.49981	.49981	.49982	.49983	.49983
3.6	.49984	.49985	.49985	.49986	.49986	.49987	.49987	.49988	.49988	.49989
3.7	.49989	.49990	.49990	.49990	.49991	.49991	.49991	.49992	.49992	.49992
3.8	.49993	.49993	.49993	.49994	.49994	.49994	.49994	.49995	.49995	.49995
3.9	.49995	.49995	.49995	.49996	.49996	.49996	.49996	.49996	.49997	.49997

Bibliography

[1] F. Poloni, *Diversi approcci al problema della classificazione.*
Online documentation.
`http://fph.altervista.org/study/files/svm.pdf`

[2] M. Sassi, A. Grissa, *Clustering large data sets based on data compression technique and weighted quality measures.*
Proceedings of the 18th international conference on Fuzzy Systems, pagg 396-402, 2009.

[3] IBM website.
`http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

[4] M. Stone *Cross-validatory choice and assessment of statistical predictions.*
Journal of the Royal Statistical Society, B, 36(1), pagg 111-147, 1974.

[5] T. Joachims *SVM^{light} Support Vector Machine.*
Online documentation.
`http://svmlight.joachims.org/`

[6] G.C. Cawley, N.L.C. Talbot, *Efficient leave-one-out cross-validation of kernel Fisher discriminant classifiers.*
The journal of the Pattern Recognition Security, 36, pagg 2585-2592, 2003.

[7] T.S.Furey, N. Cristianini, N. Duffy, D.W. Bednarski, M. Schummer, D.Haussler, *Support vector machine classification and validation of cancer tissue samples using microarray expression data.*
Bioinformatics6(10), pagg 906-914, 2000.

[8] N. Stanevski, D. Tsvetkov, *Using Support Vector Machine as a Binary Classifier.*
International Conference on Computer Systems and Technologies, pagg 1-14, 2005.

[9] S. Tong, D. Koller, *Support Vector Machine Active Learning with Applications to Text Classication*
Journal of Machine Learning Research, pagg 45-66, 2001.

[10] N. Cristianini, J. Shawe-Taylor, *An introduction to Support Vector Machines and other kernel-based learning methods.*
Cambridge University Press, 2000.

[11] I. Guyon, J. Weston, S. Barnhill, *Gene Selection for Cancer Classification using Support Vector Machines*
Machine Learning, 46, pagg 389-422, 2002.

[12] D. Du, L. Qi, R.S. Womersley, *Recent Advances in Nonsmooth Optimization*
World Scientific, pag 99, 1995.

[13] B. Schölkopf, A.J. Smola, *Learning with Kernels, Support Vector Machines, Regularization, Optimization, and Beyond.*
Library of Congress Cataloging-in-Publication Data, 2002.

[14] E. Carrizosa, D. Romero Morales, *Supervised classication and mathematical optimization.*
Computers & Operations Research, pagg 1-16, 2012.

[15] C.J.C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition.*
Data Mining and Knowledge Discovery, 2, pagg 121-167, 1998.

[16] R. Herbrich, J. Weston *Adaptive Margin Support Vector Machines for Classification.*
In Proceedings of the Ninth International Conference on Articial Neural Networks, pagg 8805, 1999.

[17] A. Moschitti, R. Basili, *VC-dimension.*
Academic material.
`http://www.uniroma2.it/didattica/WmIR/deposito/005_vc-dim.pdf`

[18] E. Torrenti, *Riconoscimento di facce in immagini generiche.*
Academic material.

[19] E. Ottaviani *Tecniche di riconoscimento statistico.* Academic material.

[20] SVM^{light} website.
`http://svmlight.joachims.org/`

[21] UCI Learning Machine Repository website.
`http://archive.ics.uci.edu/ml/`

[22] J.P. Brooks *Support Vector Machines with the Ramp Loss and the Hard Margin Loss*
Operation Research, pagg 1-13, 2011.

[23] LIBSVM website.
`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[24] R website.
`http://www.r-project.org/`

[25] A. Azzalini, B. Scarpa, *Analisi dei dati e data mining.*
Springer, 2004.

[26] D. Salvagnin, *Cenni di Programmazione Lineare Intera.*
Academic material.

[27] L. Lamport, *LaTeX, A Document Preparation System.*
Addison-Wesley Publishing Company, 1985.

[28] M. Goossens, S. Rahtz, F. Mittelbach, *The LaTeX Graphics Companion.*
Addison-Wesley Publishing Company, 1997.

[29] Ubuntu website.
`http://www.ubuntu.com/`

[30] Debianizzati website.
`http://guide.debianizzati.org/index.php/Guida_ai_comandi_da_terminale`

Acknowledgements

Inizio i miei sinceri ringraziamenti dal mio relatore, prof. ***Matteo Fischetti***.

Estremamente disponibile e gentile, ha saputo sopportare le mie mille mail, senza spazientirsi di fronte ai miei innumerevoli scoraggiamenti, ma, anzi, sapendomi sottilmente e costantemente motivare.

Grazie per la fiducia ripostami, spero di non avere deluso le sue aspettative.

Grazie innanzitutto alla mia famiglia, che c'è sempre stata, c'è sempre e ci sarà sempre. Se sono vicina o se sono lontana. Se sono felice o se sono triste.

Grazie al mio ***papi*** preferito: sai coccolarmi senza viziarmi e, con un abbraccio e poche parole, mi fai sentire capita e protetta come nessun altro.

Grazie alla mia ***mami***: nonostante tutto mi sei sempre accanto ed esprimi per me, con più speranza di ogni altra persona, il desiderio più importante: la mia felicità.

Grazie a mia sorella ***Anna***: finalmente, crescendo, ti ho riscoperta e tu, spero, hai riscoperto me. Ti voglio bene.

Grazie ai miei amici. Non mostro spesso quanto siate importanti per me, pertanto voglio cogliere questa occasione per ricordarvelo.

Grazie ad ***Ale***, per tutto quello (tanto) che abbiamo condiviso, dalle vacanze alle gite all'orto botanico, dai weekend a casa tua post erasmus alle serate "in scatola" a casa mia, dai miei costanti tentativi di distrarti a lezione alla tua pazienza nel farmi compagnia in stazione a Mestre mentre attendevo la coincidenza. Sono felice che tu ci sia nella mia vita.

Grazie ad ***Alvi***, per la tua immensa pazienza e disponibilità, perché sei una delle poche persone che ha accettato tante delle mie fragilità, per le quali provvedi a rassicurarmi costantemente. Sei una gran persona, che si fa fatica a scoprire, ma per la quale quella fatica vale proprio la pena. E "*Ace*" per un certo disegnino di pag 45 :)

Grazie ad ***Ari***, per i tuoi consigli e la tua saggezza, ma soprattutto perché ogni chiacchierata dopo tanto tempo è sempre ugualmente speciale.

Grazie a ***Bea***, per le bilaterali confidenze.

Grazie a ***Bedo***, perché sei il tipico amico che è presente nella vita di una persona nelle dosi perfette, non troppo cozzo, non troppo distante, ti interessi a me ma non sei invadente (a parte quando ti sogno durante gli incubi oppure quando vuoi infilarmi un obiettivo in un occhio!).

Grazie a ***Daniele***, per aver sopportato le mie mille domande su Ubuntu e i miei innumerevoli sfoghi, e perché.... a sminuire sono bravi tutti! :P

Grazie a ***Davide***, per credere in me.

Grazie a ***Dile***, per essere la mia criminale preferita (che sa dirmi precisamente che reato stiamo commettendo e qual è la relativa pena, in giorni ed euro) e per essere sempre così felice di sentirmi e vedermi, come lo sono io di sentire e vedere te d'altronde!

Grazie a ***Fede***, perché mi vuoi bene senza dover necessariamente stare al centro del palco, e perché sei saggia per due, sia per te che per me! Anche noi abbiamo condiviso tanto, ci siamo fatte spalla a vicenda, abbiamo riso assieme e pianto assieme (tu un bel po' meno di me però ;)). Spero con il cuore che continueremo a spartire una fettina di vita, perché adoro svegliarmi con un bradipo accanto la mattina.

Grazie a ***Fra***, perché c'è sempre e vede sempre il buono in tutto; per i messaggi che mi ricordano di non scordare che mi pensa sempre e mi augura in ogni momento un sorriso.

Grazie a ***Giulia***, per le sue chiamate "ma ciao!" tanto inaspettate quanto gradite.

Grazie a ***Giulio***, perché al suo posto non so come avrei fatto, e per fortuna lui c'è riuscito.

Grazie a ***Lorenzo***, per aver accettato la sfida: mo' so' c***i tuoi!

Grazie a ***Lory***, perché ai miei avvenimenti importanti tu non manchi mai, e non immagini quanto sia bello per me che tu ci sia.

Grazie a ***Marco***, perché ci provi SEMPRE e la cosa mi fa divertire da matti (a te un po' meno, I know).

Grazie a ***Marti***, per i numerosi consigli che hai sfornato per me, assieme ad un gregge di coccinelle! (ps: il Marco di prima non è il tuo)

Grazie a ***Miki***, per la perseveranza nello starmi vicino. E per essere incapponitamente testardo come me.

Grazie a ***Nicole***, perché sei a-d-o-r-a-b-i-l-e.

Grazie ad ***Oriez***, per le belle montagne passate assieme, sperando di replicare.

Grazie a ***Ross***, perché, anche se spesso tu ed io non siamo compatibili, sappiamo comunque riconciliarci e volerci un gran bene a prescindere.

Grazie a ***Seba***, perché nonostante tu sia burbero e lunatico, e io sia precisina e paranoica, in fondo siamo una bella coppia di amici e sappiamo equilibratamente capirci ed apprezzarci, sfotterci e mandarci a quel paese. Ridendo sempre un sacco :)

Grazie a ***Silvia***, per essere un serbatoio di malizia.

Grazie a ***Teo***, per chiamarmi con tutti quei nomignoli carini che mi sanno ringal-

luzzire l'autostima, per dedicare spesso il tuo tempo a rendermi partecipe della tua vita quotidiana, facendomi sentire come una gradita e importante ospite del tuo spettacolo, per volermi sempre sempre bene e non dimenticare mai di ricordarmelo.

Grazie poi a ***Dario***, ***Andrea*** e ***Manu*** per le serate in scatola.

Grazie a ***Bob*** perché tiene molto a me.

Grazie a ***NM***, per tutto il lavoro fatto in questi anni, anche se tanto temo gli resti da fare, perché non sa usare il computer -anche se ha un MAC- e perché si mette la crema per le mani "da fighetta".

Grazie a ***Paola***, per aver mantenuto i rapporti dopo il liceo e perché ora posso darti del tu.

Grazie ad ***Annalisa***, perché è sempre bello vederti.

Grazie a ***Mat***, per essere un cugino tanto introverso e timido quanto caro.

Varie ed eventuali....

Grazie a quell'essere peloso, scodinzolante, puzzolente, testardo e ronfante che gira per la casa e mi fa compagnia mentre lavoro. Pronta per Camilla? ;)

Padova, 10 dicembre 2012.
Luci