UNIVERSITÀ DEGLI STUDI
DI MILANO

DOTTORATO IN INFORMATICA
XXII CICLO

Classifiers based on a New Approach to Estimate the Fisher Subspace and Their Applications

Tesi di Dottorato di Ricerca di:
Alessandro Rozza

Relatore:
Prof.ssa Paola Campadelli

Correlatore:
Prof. Danilo Bruschi

Coordinatore del Dottorato:
Prof. Ernesto Damiani

Anno Accademico 2009/10

Aknowledgements

I wish to thank you...

Paola Campadelli and Danilo Bruschi for their invaluable support;

Elena Casiraghi, Gabriele Lombardi, and Stefano Arca for their pleasant presence,
and for aid that they gave me;

my parents for their support;

my girlfriend Carolina Saporiti that I love;

my family,
the lighthouse in the storm.

Dedicated to my grandfather.

Contents

Chapter 1

Introduction

Machine learning (`Ml`) is concerned with the design and development of algorithms allowing computers to learn to recognize patterns and make intelligent decisions based on empirical data. In the last few decades `Ml` has been widely investigated since it provides a general framework to build efficient algorithms solving complex problems in various application areas. The strength of machine learning algorithms relies in their capability to automatically improve their performance through experience [68], that is through data analysis methods that can be grouped into three main classes, called **supervised learning**, **unsupervised learning**, and **reinforcement learning**.

The first class of methods, which is the main focus of this thesis, comprises the **supervised learning** algorithms. These algorithms are based on a learning approach that tunes the parameters of an adaptive model by exploiting a set of N **training** points in $\Re^{D+B}$, generally represented by vectors. More precisely, each training point is composed both by an input vector (also called feature vector or instance), $\boldsymbol{x} \in \Re^D$, and its desired output vector, $\boldsymbol{t} \in \Re^B$, also called **target** or **ground-truth**.
A supervised `Ml` algorithm is applied on the training set to learn the relation between the input points and their targets; indeed, the result of this training, or learning, phase is a tuned algorithm, also called model, that implements the function $\hat{\boldsymbol{t}} = y(\boldsymbol{x}),\ \boldsymbol{x} \in \Re^D,\ \hat{\boldsymbol{t}} \in \Re^B$; this function processes an input point and generates a prediction of the output vector, which is

encoded in the same way as the target vectors, and should be as similar as possible to the desired output. The trained algorithm can then process **test points**, that are unknown input points whose ground-truth is not available during training, to predict the output $\hat{\boldsymbol{t}}$. The ability of computing the correct output when the input points differ from the training points is known as **generalization**, and it is an important characteristic that must be taken into account when assessing the performance of a supervised learning technique.

The problems handled by supervised learning methods can be divided into two classes, **classification** and **regression**. **Classification problems** need to find the association between an input point and one of finite number of discrete values, also called **labels**, that usually represent categories (classes). In this case, the `Ml` algorithms are called **classifiers**[1] and the training points are said to be labeled. In the most common applications the classes to be discriminated are considered as disjoint, so that each input point is belonging to just one class. In such cases, the input space is partitioned into **decision regions** separated by **decision boundaries** or **decision surfaces**. When these decision boundaries are $(D-1)$-dimensional linear surfaces (hyperplanes) within the D-dimensional input space, and they completely separate the classes, the classes are said to be **linearly separable**. Accordingly, **linear classifiers** are learning machines employing a linear discriminant function, that is a decision surface obtained as a linear function of the input vector $\boldsymbol{x}$.

The goal of **regression**, which is beyond the scope of this thesis, is to predict the value of one or more continuous dependent variables, $\boldsymbol{t}$, given the value of a D-dimensional independent input variable, $\boldsymbol{x}$. More specifically, application of regression analysis are generally used to predict the values of dependent variables when any one of the independent variables is varied, while the other independent variables are kept fixed.

The second class of `Ml` techniques comprises the **unsupervised learning** algorithms [30], that process input vectors without any corresponding

[1] In the following we will refer to binary, or multiclass, classifiers when the classes to be discriminated are, respectively, two or more.

target values. These techniques can be divided in two categories: **clustering**, which aims at discovering groups of 'similar' examples within the data (where the similarity criteria is depending on the employed learning algorithm), and **density estimation**, which determines the distribution that underlines the data in the input space. However, unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data.

The third class of `Ml` methods is composed of **reinforcement learning** [90] algorithms. Reinforcement learning is the problem faced by an agent (the learning algorithm) that must learn its behavior through trial-and-error interactions with a dynamic environment, in order to maximize a reward. In contrast to supervised learning, the learning algorithm is not given examples of optimal outputs but must instead discover them by a process of trial and error. Usually there is a sequence of states and actions in which the learning algorithm is interacting with its environment, and in many cases the current action affects not only the immediate reward but also the rewards of the subsequent steps.

All the above mentioned approaches are strongly dependent on the input data representation. This is the reason why several `Ml` techniques are applied on **preprocessed** input points, where the preprocessing step aims at transforming the input variables into a new space that simplifies the learning task to be solved. As an example, when a classification problem must be solved, data preprocessing is performed in order to project the data onto a new space where the classes can be better discriminated. The preprocessing stage is often referred as **feature extraction**. We note that unknown test data must be preprocessed in the same way as the training data. Feature extraction might also be performed in order to speed up the computation by decreasing the dimensionality of the input space; note that, this step must be developed in order to minimize the information loss caused by the deletion of some dimensions.

Many practical and theoretical limits that could reduce the classification

performance are still open. At first, several classification problems have at their disposal a limited percentage of training data; in this case, classifiers with an high generalization capability are needed to avoid a strong decrease of the classification accuracy during the testing phase.
On the other side, when the data are encoded in high dimensional spaces, many techniques cannot be applied for their high computational space and complexity; furthermore, when the space dimensionality is higher than the number of available training data, several algorithms become untractable since the underlying mathematical formulations become inconsistent. A similar situation could also happen when the cardinality of the training set is approximately equal to the space dimensionality.
Other problems affecting the classifier performance may be due to unbalanced training sets, i.e. training sets whose number of examples per class is strongly different.
Moreover, several learning techniques are based on strong theoretical assumptions regarding the data distribution, that might decrease the classifier robustness.
Furthermore, classifiers developed to cope with non-linearly separable classes often suffer of **overfitting** problems. More precisely, in the process of overfitting the performance of the learner on the training examples increases while the performance on unknown data becomes worse.
Finally, in multiclass classification a wide range of problems, including the need to minimize the computational complexity maintaing high accuracy, are still open.

This thesis is focused on supervised learning algorithms solving classification problems. More precisely, it describes novel, efficient, and effective classifiers overcoming the problems described above that affect the performance of existing supervised learning techniques.

To assess the performance, and to highlight the capabilities of the classifiers described in this work, we applied them on hard classification tasks, also employing high-dimensional and strongly unbalanced datasets. A further proof of the classifiers' effectiveness has been provided by the comparison of the achieved results with those computed by well-known state of the

art methods.

1.1 Outline of the Work

In this thesis, after an overview of well-known classification methods, we start by introducing our base classifier, which is a linear binary classifier based on a different approach to estimate the Fisher subspace, and its linear improvements, which deal with high dimensional data and data dynamically supplied. In the second part of the thesis we explain two different techniques to overcome the linear separability constraint thus dealing with non linearly separable classes. Besides, we present an ensemble method that effectively deals with a difficult biological multiclass classification problem. Finally, future works and conclusions are presented to the readers.

More specifically, the thesis is organized as follows:

In Chapter 2 well-known learning techniques and some mathematical and geometrical tools used in the rest of the thesis are defined.

In Chapter 3 we report the description of the Isotropic Principal Component Analysis Classifier (`IPCAC`), showing its advantages with respect to existing methods, and a technique that merges different `IPCAC` models trained on different training subsets. The effectiveness of our methods is demonstrated by experiments on synthetic data and on a spam classification task, and by comparing the computed results with those achieved by state of the art classifiers.

In Chapter 4 we present `O-IPCAC`, an online version of `IPCAC` that deals with high dimensional data, strongly unbalanced and characterized by a training set whose cardinality is approximately equal to the feature space dimensionality. This method has been compared with well-known batch and online algorithms to evaluate its efficacy.

In Chapter 5 we describe two different approaches to generalize our method to non linear classification. The first approach, called `K-IPCAC`, exploits the "kernel trick", while the second one employs a multilayer

network architecture to combine trained classifiers by exploiting a parameterless approach.

In Chapter 6 we present an ensemble method combining several `K-IPCAC`s. This method achieves promising results when it is applied to protein subcellular localization. Note that this is a particularly challenging multiclass classification problem since it is composed by 22 strongly unbalanced classes.

In Chapter 7 the summary, conclusions, and future works are presented.

Chapter 2

Overview

This chapter provides an overview about some well-known classification methods and useful geometrical and mathematical tools employed in the rest of the thesis. More precisely this chapter is organized as follows: the overview of well-known learning techniques is proposed in Section 2.1; in Section 2.2 the mathematical and geometrical tools are defined.

2.1 Background on Classification

In this section we present a brief overview of well-known classification algorithms, that have been used in several applications due to their promising results. For this reason, most of them have been employed to perform the base-line comparisons that allowed to evaluate the quality of the classifiers proposed in this thesis.

Classifiers based on Fisher Linear Discriminant Analysis
Classifiers employing the Fisher Linear Discriminant Analysis (`FLDA`) and its modified versions are linear discriminant techniques that perform classification of points projected on a subspace that maximizes the separability between classes, while minimizing the separation within each class. This subspace, called the Fisher subspace (`Fs`), is evaluated on the training data. The `Fs` and the `FLDA` approach are described in details in Sections (3.1,3.2).

Support Vector Machines
Support Vector Machines (`SVM`) [86, 95, 69] use discriminant hyperplanes to

separate points belonging to two different classes. The selected hyperplane is the one that maximizes the margins, that is the hyperplane such that the distance to the nearest data point on each side is maximized. More precisely, given a data set $\mathcal{Z} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\boldsymbol{x}_i \in \Re^D, y_i \in \mathcal{Y} = \{-1, 1\}$, where y_i are the labels of two different classes of examples, a linear classifier computes a decision function $g(\boldsymbol{x}) = \text{sign}\,(f(\boldsymbol{x}))$, where $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$.

For a point $\boldsymbol{x}_p$ on the separating hyperplane $f(\boldsymbol{x}_p) = \boldsymbol{w} \cdot \boldsymbol{x}_p + b = 0$

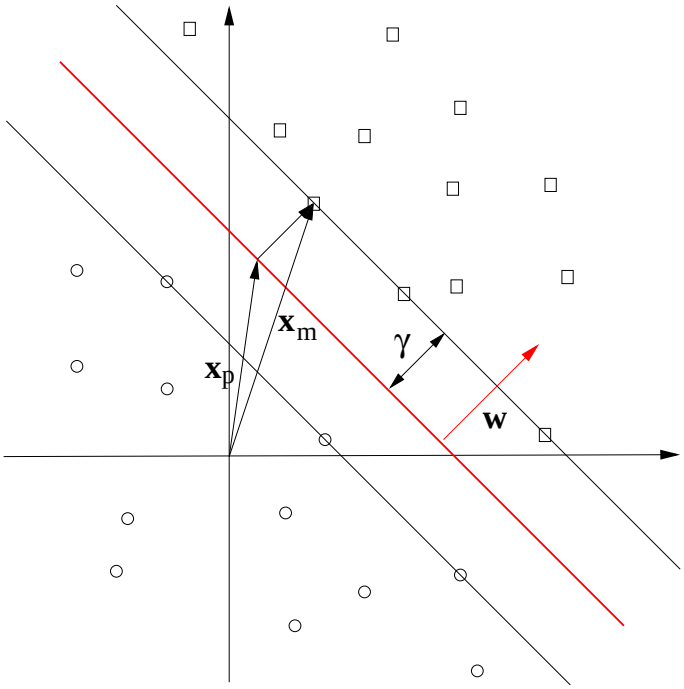


Figure 2.1: *Separating hyperplane and margins in a two-class classification problem*

(See Figure 2.1), a point $\boldsymbol{x}_m$ on the margin whose width is γ can be expressed as:

$$\boldsymbol{x}_m = \boldsymbol{x}_p + \frac{\boldsymbol{w}}{||\boldsymbol{w}||}\gamma$$

Then $f(\boldsymbol{x}_m) = \boldsymbol{w} \cdot \boldsymbol{x}_m + b = \boldsymbol{w} \cdot \boldsymbol{x}_p + \frac{\boldsymbol{w} \cdot \boldsymbol{w}}{||\boldsymbol{w}||}\gamma + b = \gamma||\boldsymbol{w}||$. The **functional margin** is $\gamma||\boldsymbol{w}||$ and the *geometric margin* is $\gamma = \frac{f(\boldsymbol{x}_m)}{||\boldsymbol{w}||}$.
To obtain the **canonical separating hyperplane** we need to normalize

w.r.t the functional margin:

$$f_c(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{\gamma||\boldsymbol{w}||}$$

The **canonical functional margin** is

$$f_c(\boldsymbol{x}_m) = \frac{f(\boldsymbol{x}_m)}{\gamma||\boldsymbol{w}||} = 1$$

The **canonical margin** is $\gamma_c = \frac{1}{||\boldsymbol{w}||}$.
From this point we consider only the canonical hyperplane (that is the hyperplane with canonical margin $1/||\boldsymbol{w}||$).
In order to maximize the margin $\gamma = \frac{1}{||\boldsymbol{w}||}$ and to correctly separate the examples we need to solve a constrained quadratic optimization problem:

$$\begin{array}{ll} \text{Minimize} & \boldsymbol{w} \cdot \boldsymbol{w} \\ \text{subject to} & y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \\ & 1 \leq i \leq n \end{array}$$

The hyperplane $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ that solves this quadratic optimization problem is the **maximal margin hyperplane** with margin $\gamma = \frac{1}{||\boldsymbol{w}||}$.
The Lagrangian associated with the primal optimization problem is:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i (y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1)$$

leading to this set of optimality conditions:

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x}_i = \boldsymbol{0} \\ \frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} &= \sum_{i=1}^{n} y_i \alpha_i = 0 \end{aligned}$$

hence

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x}_i \\ 0 &= \sum_{i=1}^{n} y_i \alpha_i \end{aligned}$$

Putting the relations obtained into the primal we have:

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} - \sum_{i=1}^{n}\alpha_i(y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1) \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) + \sum_{i=1}^{n}\alpha_i \\ &= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) \end{aligned}$$

obtaining the associated dual optimization problem:

$$\begin{array}{ll} \text{Maximize} & \Phi(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) \\ \text{subject to} & \sum_{i=1}^{n} y_i \alpha_i = 0 \\ & \alpha_i \geq 0, \quad 1 \leq i \leq n \end{array}$$

The hyperplane whose weight vector $\boldsymbol{w}^* = \sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x}_i$ solves this quadratic optimization problem is the **maximal margin hyperplane** with geometric margin $\gamma = \frac{1}{||\boldsymbol{w}||}$.
The linear SVMs compute the family of linear functions:

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{w}, b) = \{\boldsymbol{x}\cdot\boldsymbol{w} + b, \boldsymbol{w} \in \Re^D, b \in \Re\}$$

If $\boldsymbol{\alpha}^*$ is the solution of the dual optimization problem then

- $\boldsymbol{w}^* = \sum_{i=1}^{n} y_i \alpha_i^* \boldsymbol{x}_i$ is the weight vector of the maximal margin hyperplane
- $f(\boldsymbol{x}) = \boldsymbol{w}^* \cdot \boldsymbol{x} + b^* = \sum_{i=1}^{n} y_i \alpha_i^* \boldsymbol{x}_i \cdot \boldsymbol{x} + b^*$ is the corresponding discriminant function.
- The decision function $g : \Re^D \rightarrow \{-1, +1\}$ is $g(\boldsymbol{x}) = \text{sign}(\sum_{i=1}^{n} y_i \alpha_i^* \boldsymbol{x}_i \cdot \boldsymbol{x} + b^*)$

The SVM algorithm minimizes both the empirical risk and the confidence interval [94]. Indeed, maximizing the margin, that is equivalently minimizing $||\boldsymbol{w}||$, we minimize the Vapnik Chervonenkis (VC) dimension, and the confidence interval depends mainly on the ratio (VC) dimension/cardinality of the training set.
Nevertheless, the SVM defined above deals with linearly separable data. In order to consider non linearly separable data we need to introduce soft margin SVM and kernels. In this setting we first add to the primal optimization

problems a set of slack variables ξ_i and their corresponding constraints, thus obtaining:

$$\begin{array}{ll} \text{Minimize} & \boldsymbol{w} \cdot \boldsymbol{w} + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} & y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & 1 \leq i \leq n \end{array}$$

If $K(\boldsymbol{x}, \boldsymbol{x}')$ is a symmetric function satisfying **Mercer's conditions**, that is:

$$\int \int K(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}) f(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{x}' \geq 0$$

for all f such that $\int f^2(\boldsymbol{x}) d\boldsymbol{x} < \infty$, then we can expand $K(\boldsymbol{x}, \boldsymbol{x}')$ in a some inner product feature space:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j=1}^{\infty} \lambda_j \phi(\boldsymbol{x}) \phi(\boldsymbol{x}')$$

Note that in the dual representation of linear `SVM`s the inputs appears only in a dot-product form; as a consequence, we can substitute the dot-products in the input space with a kernel function obeying Mercer's conditions:

$$\begin{array}{ll} \text{Maximize} & \Phi(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i \boldsymbol{x}_j) \\ \text{subject to} & \sum_{i=1}^{n} y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad 1 \leq i \leq n \end{array}$$

The discriminant function computed by , and obtained from the solution of this quadratic optimization problem, is:

$$f(\boldsymbol{x}, \alpha^*, b) = \sum_{i=1}^{n} y_i \alpha_i^* K(\boldsymbol{x}_i, \boldsymbol{x}) + b^*$$

In this case, the input patterns, $\boldsymbol{x}$, processed by the `Kernel SVM` belong to the input space, but the `SVM` itself works in an high dimensional (possibly infinite) feature space, where it performs a linear separation of the data. The symmetric function $K(\cdot, \cdot)$ must be chosen among the kernels of Reproducing Kernel Hilbert Spaces [97]; three possible choices are:

- Linear kernel: $K(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u} \cdot \boldsymbol{v}$
- Polynomial kernel: $K(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u} \cdot \boldsymbol{v} + 1)^d$

- Gaussian kernel: $K(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\|\boldsymbol{u} - \boldsymbol{v}\|^2/\sigma^2)$

The bias and variance of SVMs are typically controlled by two parameters. Parameter C controls the tradeoff between fitting the data (achieved by driving the ξ_i's to zero) and maximizing the margin (achieved by driving $\|\boldsymbol{w}\|$ to zero). Setting C large should tend to minimize bias. The second parameter that controls bias arises only in SVMs that employ parameterized kernels such as the polynomial kernel (where the parameter is the degree d of the polynomial) and RBF kernels (where the parameter is the width σ of the gaussian kernel). Bias and variance depend critically on these parameters [93].
The SVM technique is considered to be insensitive to overfitting, and it has good generalization capabilities [53]; for this reasons it is a good base-line for performance comparison, and it has been widely employed in our tests.

Nearest Neighbours and K-Nearest Neighbors

The Nearest Neighbor classifier (NN, [26]) is a supervised learning method that classifies previously unseen examples $\boldsymbol{x}$ by calculating the distances to the training cases and then using the label of the nearest training example as the final decision.
Generally, the NN classifier employs the euclidean distance to find the nearest neighbor of an unknown test sample $\boldsymbol{x}$. Let $\boldsymbol{x}_i$ be an input sample in $\Re^D$, the euclidean distance between two samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_l$ $(i, l = 1, 2, \ldots, N)$ is defined as:

$$d(\boldsymbol{x}_i, \boldsymbol{x}_l) = \sqrt{(x_{i,1} - x_{l,1})^2 + \cdots + (x_{i,D} - x_{l,D})^2} = \|\boldsymbol{x}_i - \boldsymbol{x}_l\|_2 .$$

A graphical depiction of the nearest neighbor concept is illustrated in the Voronoi tesellation ([96], see Figure 2.2). A Voronoi cell encapsulates all the neighboring points that are nearest to each sample. Given a set of N points, $\boldsymbol{x}_i$ $(i = 1, 2, \ldots, N)$, the Voronoi cell for $\boldsymbol{x}_i$, referred as R_i, is defined as:

$$R_i = \{\boldsymbol{y} \in \Re^D : d(\boldsymbol{y}, \boldsymbol{x}_i) \leq d(\boldsymbol{y}, \boldsymbol{x}_m), \quad \forall \quad i \neq m, \quad m = 1, 2, \ldots, N\}$$

Notice that all the possible points within a sample's Voronoi cell are the nearest neighbors of that sample.

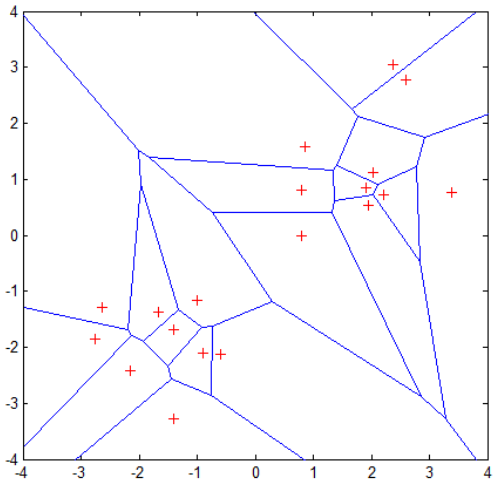


Figure 2.2: Voronoi tesellation showing Voronoi cells of 19 samples marked with a "+".

The K-Nearest Neighbors classifier (KNN, [30]) extends this idea by finding in the training set the first K nearest neighbours of $\boldsymbol{x}$, and assigning to $\boldsymbol{x}$ the majority category label of its K nearest points. In Figure 2.3 an intuitive example of KNN classification is given; considering the green circle as the test sample, it should be classified as belonging to either the first class of blue squares, or the second class of red triangles. If $K = 3$ it is classified as belonging to the second class because there are 2 triangles and only 1 square inside the inner circle, whilst if $K = 5$ it is classified as belonging to the first class.

Mahalanobis distance based classifiers

Mahalanobis distance based classifiers assume that each class is described by a Multivariate Gaussian distribution. Considering a set of clustered points $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}}_c\}_{c=1}^{C} = \{\boldsymbol{p}_i \in \Re^D\}_{i=1}^{N}$, and the class means $\boldsymbol{\mu}_c, c = 1, .., C$, an unknown point $\boldsymbol{x} \in \Re^D$ is assigned to the class of the nearest $\boldsymbol{\mu}_c$; in details, the selected class is the one that minimizes the distance function $d(\boldsymbol{x}, \boldsymbol{\mu_c})$, that is computed as the Mahalanobis distance between the point $\boldsymbol{x}$ and the mean vector of class c, $\boldsymbol{\mu_c}$. More precisely, $d(\boldsymbol{x}, \boldsymbol{\mu_c})$ is computed as follows:

$$d(\boldsymbol{x}, \boldsymbol{\mu_c}) = \sqrt{(\boldsymbol{x} - \boldsymbol{\mu}_c)\, \boldsymbol{\Sigma}_c^{-1}\, (\boldsymbol{x} - \boldsymbol{\mu}_c)^T}$$

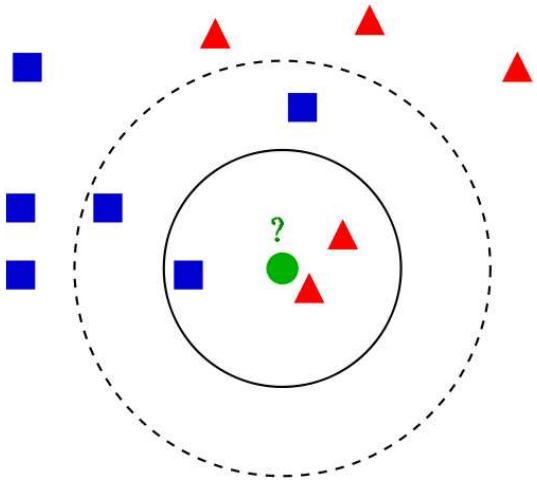


Figure 2.3: Example of KNN classification.

where $\boldsymbol{\Sigma}_c$ is the class covariance matrix:

$$\begin{aligned} \boldsymbol{\mu}_c &= \frac{1}{N_c}\sum_{i=1}^{N_c}\boldsymbol{p}_{c,i} \\ \boldsymbol{\Sigma}_c &= \frac{1}{N_c}\sum_{i=1}^{N_c}\left(\boldsymbol{p}_{c,i}-\boldsymbol{\mu}_c\right)\left(\boldsymbol{p}_{c,i}-\boldsymbol{\mu}_c\right)^T, \end{aligned}$$

where N_c is the cardinality of the class c, $\boldsymbol{p}_{c,i}$ is the i-th sample of class c.

2.2 Main Geometrical Algorithms

In this section the main geometrical algorithms used in the rest of the thesis are briefly recalled.

2.2.1 Matrix Decomposition

Given a symmetric positive semi-definite square matrix $\boldsymbol{T} \in \Re^{D\times D}$, several applications might need to diagonalize it; to this aim, the eigen-system $\boldsymbol{T} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T$ must be solved to find the set of real valued [1] eigenvectors $\boldsymbol{x}_i$, and the associated eigenvalues λ_i, such that $\forall i, \boldsymbol{T}\boldsymbol{x}_i = \lambda_i\boldsymbol{x}_i$.

Being $\boldsymbol{T}$ symmetric, the eigenvalues can be estimated using a very efficient algorithm. To understand it, we highlight that a symmetric positive

[1] Note that the eigenvectors and the eigenvalues are real-valued as long as $\boldsymbol{T}$ is positive semi-definite, otherwise they are complex-valued.

semi-definite square matrix can be geometrically interpreted as an ellipsoid, and its diagonalization extracts the orthogonal vectors (eigenvectors) oriented along the ellipsoid axes, and their length. To this aim, the principal axes can be rotated by the inverse of the eigenvector matrix $\boldsymbol{X}^T$ transforming them into vectors aligned with the canonical basis; this transformation allows to obtain the diagonal matrix $\boldsymbol{X}^T\boldsymbol{T}\boldsymbol{X} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}\boldsymbol{\Lambda}\boldsymbol{I} = \boldsymbol{\Lambda}$. Since $\boldsymbol{X}$ is not known, an iterative technique, such as the *Jacobi method*, must be used.

The Jacobi method is based on the *Givens rotation matrices*: chosing two integers p and q such that $1 \leq p < q \leq D$, an angle θ can be found so that the following orthogonal similarity holds:

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} \boldsymbol{T}^i_{p,p} & \boldsymbol{T}^i_{p,q} \\ \boldsymbol{T}^i_{q,p} & \boldsymbol{T}^i_{q,q} \end{pmatrix} \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} \boldsymbol{T}^{i+1}_{p,p} & 0 \\ 0 & \boldsymbol{T}^{i+1}_{q,q} \end{pmatrix} \tag{2.1}$$

in fact the angle $\theta = 0$ can be chosen if $\boldsymbol{T}^i_{p,q} = \boldsymbol{T}^i_{q,p} = 0$; otherwise, if $\boldsymbol{T}^i_{p,q} = \boldsymbol{T}^i_{q,p} \neq 0$, it must be $\boldsymbol{T}^{i+1}_{p,q} = \boldsymbol{T}^{i+1}_{q,p} = 0$. To this aim, the following derivations allow to compute the angle θ representing the correct rotation:

$$\begin{aligned} \boldsymbol{T}^{i+1}_{p,q} = 0 \quad &\Rightarrow \quad (\boldsymbol{T}^i_{p,p} - \boldsymbol{T}^i_{q,q})\cos(\theta)\sin(\theta) + \boldsymbol{T}^i_{p,q}(\sin^2(\theta) - \cos^2(\theta)) = 0 \\ &\Rightarrow \quad \theta = \arctan\left(\frac{\boldsymbol{T}^i_{q,q} - \boldsymbol{T}^i_{p,p}}{2\boldsymbol{T}^i_{p,q}} + \sqrt{\left(\frac{\boldsymbol{T}^i_{q,q} - \boldsymbol{T}^i_{p,p}}{2\boldsymbol{T}^i_{p,q}} \right)^2 + 1} \right)^{-1} \end{aligned} \tag{2.2}$$

Computing this transformation for all index pairs $1 \leq p < q \leq D$ allows to transform $\boldsymbol{T}$ into the diagonal eigenvalues matrix $\boldsymbol{\Lambda}$ by means of a finite sequence of at most $\frac{D(D-1)}{2}$ rotations.

To compute the associated eigenvectors, in the hypothesis that the dominant eigenvalue λ_1 has geometrical multiplicity 1, it is possible to randomly choose an initial non-zero estimate of $\boldsymbol{x}_1$, say $\boldsymbol{x}^0_1$, and employ the iterative method $\boldsymbol{x}^{i+1}_1 = \frac{\boldsymbol{T}\boldsymbol{x}^i_1}{\|\boldsymbol{T}\boldsymbol{x}^i_1\|}$ that converges to $\boldsymbol{x}_1$. Noting that it must be $\forall i,\ \boldsymbol{T}\boldsymbol{T}\boldsymbol{x}_i = \lambda_i\boldsymbol{T}\boldsymbol{x}_i = \lambda^2_i\boldsymbol{x}_i$, and in general $\boldsymbol{T}^K\boldsymbol{x}_i = \lambda^K_i\boldsymbol{x}_i$, it is possible to obtain a faster algorithm fixing a K value and computing directly $\boldsymbol{x}^{i+K}_1 = \frac{\boldsymbol{T}^K\boldsymbol{x}^i_1}{\|\boldsymbol{T}^K\boldsymbol{x}^i_1\|}$; this iterative method is the basis of the *power method* for

eigenvectors computation. This procedure can be iterated by computing the new matrix $\tilde{\boldsymbol{T}} = \boldsymbol{T} - \lambda_1 \boldsymbol{x}_1 \boldsymbol{x}_1^T$ and finding the next eigenvalue/eigenvector pair. This algorithm follows the decomposition $\boldsymbol{T} = \sum_{i=1}^{D} \lambda_i \boldsymbol{x}_i \boldsymbol{x}_i^T$ obtained by:

$$\boldsymbol{T} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T = \boldsymbol{X}\left(\sum_{i=1}^{D} \tilde{\boldsymbol{\Lambda}}_i\right)\boldsymbol{X}^T = \sum_{i=1}^{D} \boldsymbol{X}\tilde{\boldsymbol{\Lambda}}_i\boldsymbol{X}^T = \sum_{i=1}^{D} \lambda_i \boldsymbol{x}_i \boldsymbol{x}_i^T \quad (2.3)$$

On the other side, if the eigenvalues matrix $\boldsymbol{\Lambda}$ and the eigenvector matrix $\boldsymbol{X}$ are known, the matrix $\boldsymbol{T}$ can be *encoded* by simply computing $\boldsymbol{T} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T$; nevertheless, due to roundoff machine precision errors, the obtained matrix is redundant and no longer symmetric. Therefore, it is better to compute only the $\frac{D(D+1)}{2}$ values for the upper triangular part, and store them instead of the D^2 redundant coefficients.

For details, and other efficient algorithms to solve eigen-systems in case of symmetric positive semi-definite matrices, see [33].

2.2.2 Singular Value Decomposition

Given a $D \times N$ matrix $\boldsymbol{M}$ composed of real, or complex, valued vectors, it can be factorized as follows:

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T \quad (2.4)$$

where $\boldsymbol{U}$ is the the $D \times D$ unitary (left) singular vector matrix over the field membership, the matrix $\boldsymbol{Q}$ is a $D \times N$ diagonal singular valued matrix with non-negative real numbers on the diagonal, and $\boldsymbol{V}^T$ denotes the conjugate transpose of $\boldsymbol{V}$, which is an $N \times N$ unitary matrix. Such a factorization is called a singular value decomposition (`SVD`) of $\boldsymbol{M}$.

The `SVD` can be applied to any $D \times N$ matrix, while the eigenvalue-decomposition can only be applied to positive semi-definite square matrices. Nevertheless, the two decompositions are related. Indeed, considering the `SVD` of $\boldsymbol{M}$ the following two relations hold:

$$\begin{aligned} \boldsymbol{M}^T\boldsymbol{M} &= \boldsymbol{V}\boldsymbol{Q}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T = \boldsymbol{V}(\boldsymbol{Q}^T\boldsymbol{Q})\boldsymbol{V}^T, \\ \boldsymbol{M}\boldsymbol{M}^T &= \boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{Q}^T\boldsymbol{U}^T = \boldsymbol{U}(\boldsymbol{Q}\boldsymbol{Q}^T)\boldsymbol{U}^T. \end{aligned}$$

where the right hand sides of these relations describe the eigenvalue-decompositions of the left hand sides. Under these consideration the columns of $\boldsymbol{U}$ (left singular vectors) are eigenvectors of $\boldsymbol{M}\boldsymbol{M}^T$ and the columns of $\boldsymbol{V}$ (right singular vectors) are eigenvectors of $\boldsymbol{M}^T\boldsymbol{M}$.

Considering the example of covariance matrix estimation, where the covariance matrix is estimated as $\tilde{\boldsymbol{\Sigma}} = \frac{1}{N-1}\boldsymbol{M}\boldsymbol{M}^T$, then the eigen-decomposition of $\tilde{\boldsymbol{\Sigma}}$, that is $\tilde{\boldsymbol{\Sigma}} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T$ can be performed through the `SVD` of $\boldsymbol{M}$, that is $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T$, since in this case $\boldsymbol{M}\boldsymbol{M}^T = \boldsymbol{U}\boldsymbol{Q}^2\boldsymbol{U}^T$, thus obtaining $\boldsymbol{\Lambda} = \frac{1}{N-1}\boldsymbol{Q}^2$ and $\boldsymbol{X} = \boldsymbol{U}$.

2.2.3 Orthogonalization

Given a $D \times d$ matrix $\boldsymbol{V} = [\boldsymbol{v}_1|\cdots|\boldsymbol{v}_d]$ with d linear independent columns spanning a d-dimensional subspace of $\Re^D$, it might be useful to find an orthonormal basis $\boldsymbol{X}$ of $\Re^D$ so that the first d columns of $\boldsymbol{X}$ span the same subspace spanned by $\boldsymbol{V}$; this problem has infinite solutions. To find one of the possible solutions, the *Gram-Schmidt orthogonalization* method can be used:

- set $\boldsymbol{x}_1 = \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|}$;
- compute every $\boldsymbol{x}_i$ with $1 < i \leq d$ as $\boldsymbol{x}_i = \frac{\boldsymbol{v}_i - \sum_{j=1}^{i-1}(\boldsymbol{x}_j \cdot \boldsymbol{v}_i)\boldsymbol{x}_j}{\|\boldsymbol{v}_i - \sum_{j=1}^{i-1}(\boldsymbol{x}_j \cdot \boldsymbol{v}_i)\boldsymbol{x}_j\|}$;
- eventually, compute the remaining $D - d$ orthonormal vectors $\boldsymbol{x}_i$ with $d < i \leq D$, to span the whole space $\Re^D$. This can be done as in the previous step by choosing every time an initial vector $\boldsymbol{v}_i$ linearly independent with respect to $\boldsymbol{x}_j$ for $1 \leq j < i$. A simple way to get a vector $\boldsymbol{v}_i$ linearly independent w.r.t. the vectors $\boldsymbol{v}_{j<i}$ is to generate it randomly.

The first d orthonormal vectors obtained by this algorithm span the same subspace spanned by the $\boldsymbol{v}_i$, while the other $D - d$ orthogonal vectors span its normal subspace.

2.2.4 Principal Component Analysis (`PCA`)

Given a set of points $\boldsymbol{\mathcal{P}} = \{\boldsymbol{p}_i \in \Re^D\}_{i=1}^N$, drawn from an unknown Gaussian distribution, the **Principal Component Analysis** (`PCA`) transform

is aimed at projecting the points on a d-dimensional space ($d \leq D$), such that the variance of the projected data is maximized. When $d < D$, PCA is used as a convenient dimensionality reduction method that reduces the loss of discriminative information. The idea is to find the directions in $\Re^D$ where the variance of the data is greater, so that the input points can be approximated by representing them with a basis that spans only a linear subspace excluding the low variance directions.

To this aim, the following maximum likelihood estimators are used:

$$\begin{aligned} \tilde{\boldsymbol{\mu}} &= \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{p}_i \\ \tilde{\boldsymbol{\Sigma}} &= \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})(\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})^T \end{aligned}$$

Performing the eigen-decomposition $\tilde{\boldsymbol{\Sigma}} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T$, and maintaining the eigenvalues/eigenvectors sorted in descendant eigenvalues order, the obtained first d column-vectors $\boldsymbol{x}_j$ of $\boldsymbol{X}$ ($j = 1, .., d$) represent an orthonormal basis for the d-dimensional linear subspace that "captures" the highest variance of the points in $\boldsymbol{\mathcal{P}}$. Representing with $\boldsymbol{X}_d$ the selected basis, the point projections on this subspace can be obtained by means of equation:

$$\boldsymbol{p}_\perp = \boldsymbol{X}_d^T(\boldsymbol{p} - \tilde{\boldsymbol{\mu}})$$

where the d values $\boldsymbol{p}_\perp$ are called *PCA coefficients*. On the other side, given a coefficient vector $\boldsymbol{b}$, the corresponding point $\boldsymbol{p} \in \Re^D$ can be computed as:

$$\boldsymbol{p} = \boldsymbol{X}_d\boldsymbol{b} + \tilde{\boldsymbol{\mu}}$$

2.2.5 Whitening Process

The process of data whitening is a geometrical approach that moves the data in an isotropic position. More precisely, after employing this process the points are mean centered and the data covariance matrix equals the identity matrix; thus, from a geometrical point of view, the points assume the shape of an hypersphere centered in the origin. Considering the set of training points $\hat{\boldsymbol{\mathcal{P}}} = \bigcup_{c=1}^{C}\boldsymbol{\mathcal{P}}_c = \{\boldsymbol{p}_i\}_{i=1}^{N}$, where c is the class membership, the whitening matrix $\boldsymbol{W}$ is estimated as follows:

1. estimate the expectation $\tilde{\boldsymbol{\mu}} = \frac{1}{N}\sum_i \boldsymbol{p}_i$, and the covariance matrix $\tilde{\boldsymbol{\Sigma}} = \frac{1}{N-1}\sum_i (\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})(\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})^T$;

2. estimate the principal components through the eigen-decomposition of the covariance matrix $\boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T = \tilde{\boldsymbol{\Sigma}}$;

3. estimate the whitening matrix as $\boldsymbol{W} = \boldsymbol{X}\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{X}^T$. Note that $\boldsymbol{\Lambda}^{-\frac{1}{2}}$ can be computed by substituting the non-zero diagonal elements λ_i of $\boldsymbol{\Lambda}$ with the values $\lambda_i^{-\frac{1}{2}}$.

Chapter 3

Isotropic Principal Component Analysis-based Classifier

In this chapter we propose a linear binary classifier, called Isotropic Principal Component Analysis-based Classifier (`IPCAC`), that deals with data points drawn from a Mixture of Gaussians (`MoG`). `IPCAC` can be considered as an evolution of the Fisher Linear Discriminant Analysis (`FLDA`) that employs a novel approach to estimate the Fisher subspace. More precisely, the estimation of this subspace is based on the theoretical results, reported by Brubaker and Vempala in [10], proving that, given an isotropic `MoG`, the Fisher subspace underlying the data corresponds to the span of the Gaussian means.

This chapter is organized as follows: in Section 3.1 the definition of the Fisher subspace is reported; in Section 3.2 the Fisher Linear Discriminant Analysis technique is summarized; in Section 3.3 we consider data points distributed according to an isotropic mixture, and we show that the Fisher subspace equals the intermean subspace; in Section 3.4 the `IPCAC` classifier is presented, and its advantages are highlighted; in Section 3.5 experimental results are reported, together with the performance comparison between `IPCAC` and `FLDA`; in Section 3.6 a model merging technique to combine different `IPCAC`s is described; in Section 3.7 the efficacy of the proposed methods is demonstrated by their application on spam classification, and by comparing

their results with those achieved by well-known techniques.

3.1 Fisher SubSpace

Consider a set of clustered points $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}}_c\}_{c=1}^{C}$ drawn from $\Re^D$, where each cluster $\boldsymbol{\mathcal{P}}_c = \{\boldsymbol{p}_{c,i}\}_{i=1}^{N_c}$ contains N_c points (feature vectors). In [4], it is proved that it is possible to find a $(C-1)$-dimensional linear subspace $\boldsymbol{Fs}_{\boldsymbol{\mathcal{P}}}$, called the Fisher subspace, defined by the given point set $\boldsymbol{\mathcal{P}}$, that maximizes the linear separability among the classes. More precisely, the subspace $\boldsymbol{Fs}_{\boldsymbol{\mathcal{P}}}$ is presented as the subspace that maximizes a function, called the Fisher's criterion, that produces a large separation between the projected classes while simultaneously minimizing the variance within each projected class. The Fisher's criterion is:

$$\boldsymbol{Fs}_{\boldsymbol{\mathcal{P}}} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, J(\boldsymbol{w}) \quad = \quad \underset{\boldsymbol{w}}{\operatorname{argmax}} \frac{\text{between-class variance}(\boldsymbol{\mathcal{P}} \text{ proj on } \boldsymbol{w})}{\text{within-class variance}(\boldsymbol{\mathcal{P}} \text{ proj on } \boldsymbol{w})}$$

In [29, 38] the $\boldsymbol{Fs}_{\boldsymbol{\mathcal{P}}}$ is also defined as the maximization of the following discriminant function:

$$\begin{aligned} \boldsymbol{Fs}_{\boldsymbol{\mathcal{P}}} = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, J(\boldsymbol{w}) &= \underset{\boldsymbol{w}}{\operatorname{argmax}} \frac{\text{total variance}}{\text{within-class variance}(\boldsymbol{\mathcal{P}} \text{ proj on } \boldsymbol{w})} \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}} \frac{\mathbb{E}_{c,i}\left[\|proj_{\boldsymbol{w}}(\boldsymbol{p}_{c,i} - \boldsymbol{\mu})\|^2\right]}{\mathbb{E}_c\left[\mathbb{E}_i\left[\|proj_{\boldsymbol{w}}(\boldsymbol{p}_{c,i} - \boldsymbol{\mu}_c)\|^2\right]\right]} \end{aligned}$$

where i indexes the points in each cluster, $proj_{\boldsymbol{w}}(\cdot)$ is the linear operator that projects a point on the subspace $\boldsymbol{w}$, $\mathbb{E}_{\cdot}[\cdot]$ is the expectation operator ($\mathbb{E}_{c,i}[\cdot]$ is the expectation over all the training set, $\mathbb{E}_i[\cdot]$ is the expectation over the c^{th} class, and $\mathbb{E}_c[\cdot]$ is the expectation over all the classes), $\boldsymbol{\mu}$ and $\boldsymbol{\mu}_c$ are respectively the overall mean and the c-cluster mean. In the following section we report the rationale at the basis of the Fisher's criterion, together with its mathematical formulation, and its solution.

3.2 The Fisher Linear Discriminant Analysis

The most common approach to estimate the Fisher subspace is the Fisher Linear Discriminant Analysis (FLDA, [4]) technique. To describe it we firstly consider a two class classification problem, where A and B are the two

classes to be discriminated. If the points to be classified belong to a D-dimensional space, a vector $\boldsymbol{w} \in \Re^D$ can be found so that a point $\boldsymbol{x}$ can be projected on $\boldsymbol{w}$ through $y = \boldsymbol{w}^T\boldsymbol{x}$; given a threshold value w_0, each point can be classified as belonging to class A if $y \geq w_0$, to class B otherwise. To achieve good classification results from this classifier, the goal is to find a vector $\boldsymbol{w}$, commonly called *optimal discriminant projection vector*, such that the separability of the projected classes is maximal. Simultaneously, $\boldsymbol{w}$ should be defined in order to minimize the scattering within each class. The idea proposed by Fisher is to maximize a function that produces the largest separation between the projected class means while maintaining a small variance of the projected points within each class. To formulate this function, called the Fisher's criterion, we identify with N_A and N_B the number of points belonging to class A, and to class B respectively, and we refer with $\boldsymbol{\mu}_a$ and $\boldsymbol{\mu}_b$ the class means in the original space:

$$\boldsymbol{\mu}_a = \frac{1}{N_A}\sum_{i=1}^{N_A}\boldsymbol{x}_i, \qquad \boldsymbol{\mu}_b = \frac{1}{N_B}\sum_{j=1}^{N_B}\boldsymbol{x} \tag{3.1}$$

Considering the projection vector $\boldsymbol{w}$, the separation among the projected class means is:

$$\mu_a - \mu_b = \boldsymbol{w}^T(\boldsymbol{\mu}_a - \boldsymbol{\mu}_b), \quad where \quad \mu_a = \boldsymbol{w}^T\boldsymbol{\mu}_a, \quad \mu_b = \boldsymbol{w}^T\boldsymbol{\mu}_b$$

while the variance of the projected points within each class is:

$$\sigma_a^2 = \sum_{j=1}^{N_A}(x_j - \mu_a), \quad and \quad \sigma_b^2 = \sum_{j=1}^{N_B}(x_j - \mu_b), \quad where \quad x_j = \boldsymbol{w}^T\boldsymbol{x}_j$$

The Fisher's criterion is then defined as:

$$J(\boldsymbol{w}) = \frac{(\mu_a - \mu_b)^2}{\sigma_a^2 + \sigma_b^2}$$

Alternatively, $J(\boldsymbol{w})$ can be defined in order to make the dependence from $\boldsymbol{w}$ explicit, as follows:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T\boldsymbol{\Sigma}_{Bet}\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{\Sigma}_W\boldsymbol{w}} \tag{3.2}$$

where $\boldsymbol{\Sigma}_{Bet}$ is the between-class covariance matrix defined as:

$$\boldsymbol{\Sigma}_{Bet} = (\boldsymbol{\mu}_b - \boldsymbol{\mu}_a)(\boldsymbol{\mu}_b - \boldsymbol{\mu}_a)^T \tag{3.3}$$

and $\mathbf{\Sigma}_W$ is the total within-class covariance matrix:

$$\mathbf{\Sigma}_W = \sum_{i=1}^{N_A}(\boldsymbol{x}_i - \boldsymbol{\mu}_a)(\boldsymbol{x}_i - \boldsymbol{\mu}_a)^T + \sum_{j=1}^{N_B}(\boldsymbol{x}_j - \boldsymbol{\mu}_b)(\boldsymbol{x}_j - \boldsymbol{\mu}_b)^T \tag{3.4}$$

Differentiating Equation (3.2) with respect to $\boldsymbol{w}$, we find that $J(\boldsymbol{w})$ is maximal when:

$$\left(\boldsymbol{w}^T\mathbf{\Sigma}_{Bet}\boldsymbol{w}\right)\mathbf{\Sigma}_W\boldsymbol{w} = \left(\boldsymbol{w}^T\mathbf{\Sigma}_W\boldsymbol{w}\right)\mathbf{\Sigma}_{Bet}\boldsymbol{w} \tag{3.5}$$

Considering that Σ_{Bet} is oriented along $(\boldsymbol{\mu}_b - \boldsymbol{\mu}_a)$, and that both $\left(\boldsymbol{w}^T\mathbf{\Sigma}_{Bet}\boldsymbol{w}\right)$ and $\left(\boldsymbol{w}^T\mathbf{\Sigma}_W\boldsymbol{w}\right)$ can be dropped since the magnitude of $\boldsymbol{w}$ is not important, we obtain:

$$\boldsymbol{w} \propto \mathbf{\Sigma}_W^{-1}(\boldsymbol{\mu}_b - \boldsymbol{\mu}_a). \tag{3.6}$$

The resulting vector, which could also be computed by finding the eigenvector of $\mathbf{\Sigma}_W^{-1}\mathbf{\Sigma}_{Bet}$ corresponding to the largest eigenvalue, is the direction that allows to project data points, belonging to two classes $A \in \Re^D, B \in \Re^D$, onto the one-dimensional Fisher subspace.

3.2.1 Fisher subspace for multiple classes

When C classes are considered, the points are projected on a $(C-1)$-dimensional Fisher subspace, through a projection matrix $\boldsymbol{W}$ whose columns are the optimal discriminant projection vectors $\boldsymbol{w}_k,\ k = 1, .., C-1$. The projection matrix is obtained by generalizing both the within-class and the between-class scatter matrices to the case of C classes. More precisely, the within-class covariance matrix is defined as follows:

$$\mathbf{\Sigma}_W = \sum_{c=1}^{C}\mathbf{\Sigma}_c, \qquad \mathbf{\Sigma}_c = \sum_{j=1}^{N_c}(\boldsymbol{x}_j - \boldsymbol{\mu}_c)(\boldsymbol{x}_j - \boldsymbol{\mu}_c)^T$$

To generalize the between-class scatter matrix to the C-class case, we consider the total covariance matrix:

$$\mathbf{\Sigma}_{Tot} = \sum_{j=1}^{N}(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T$$

and we notice that it can be decomposed it into the sum of $\mathbf{\Sigma}_W$ and a matrix that can be viewed as a measure of the between-class covariance

matrix, $\mathbf{\Sigma}_{Bet}$. More precisely, setting $\mathbf{\Sigma}_{Tot} = \mathbf{\Sigma}_W + \mathbf{\Sigma}_{Bet}$ we can compute $\mathbf{\Sigma}_{Bet}$ as:

$$\mathbf{\Sigma}_{Bet} = \sum_{c=1}^{C} N_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T$$

Exploiting these equations, in [38] the Fisher's criterion in the multiclass case is defined as:

$$J(\boldsymbol{W}) = Tr\left\{\frac{\boldsymbol{W}^T\mathbf{\Sigma}_{Bet}\boldsymbol{W}}{\boldsymbol{W}^T\mathbf{\Sigma}_W\boldsymbol{W}}\right\} \tag{3.7}$$

As pointed out by Fukanaga there are many choices of criterion. Indeed, in the same work another variant of this criterion is showed, where the quantity to be maximized is:

$$\bar{J}(\boldsymbol{W}) = Tr\left\{\frac{\boldsymbol{W}^T\mathbf{\Sigma}_{Tot}\boldsymbol{W}}{\boldsymbol{W}^T\mathbf{\Sigma}_W\boldsymbol{W}}\right\} \tag{3.8}$$

Similarly, in [60] a third definition of the Fisher's criterion, exploiting $\mathbf{\Sigma}_{Tot}$, is reported:

$$\hat{J}(\boldsymbol{W}) = Tr\left\{\frac{\boldsymbol{W}^T\mathbf{\Sigma}_{Bet}\boldsymbol{W}}{\boldsymbol{W}^T\mathbf{\Sigma}_{Tot}\boldsymbol{W}}\right\} \tag{3.9}$$

As described at length in [38], the set of projection vectors $\boldsymbol{w}_k,\ k = 1, .., C-1$ that allow to maximize all the above mentioned Fisher's criteria, is the set of eigenvectors of $\mathbf{\Sigma}_W^{-1}\mathbf{\Sigma}_{Bet}$ corresponding to the largest eigenvalues. For these reason $\bar{J}(\boldsymbol{W})$, $\hat{J}(\boldsymbol{W})$, and $J(\boldsymbol{W})$ are said to be functionally equivalent in terms of solving the optimal set of projection axes [60], that is:

$$\underset{\boldsymbol{W}}{\text{argmax}}\ \bar{J}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\text{argmax}}\ \hat{J}(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\text{argmax}}\ J(\boldsymbol{W})$$

After finding the best projection subspace, classification can be performed by exploiting different approaches proposed in the literature to analyze the training points projected on $\boldsymbol{w}$ in order to identify either separation hyperplanes (in case of multiclass classification) or a thresholding value (in case of binary classification). As an example, classification can be performed in the transformed space by employing some distance metric, such as the Euclidean distance ($d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_i (x_i - y_i)^2}$) or the cosine measure

($d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$), and classifying a new instance $\boldsymbol{z}$ as belonging to the class $\hat{c}$ that minimizes the following:

$$\hat{c} = \underset{c}{\operatorname{argmin}}\ d\left(\boldsymbol{w}^T\boldsymbol{z}, \boldsymbol{w}^T\boldsymbol{\mu}_c\right),$$

where $\boldsymbol{\mu}_c$ is the centroid of the training subset composed of points belonging to the c^{th} class [59].

Due to its efficacy, and simplicity, FLDA is widely employed in different application fields by several applications that exploit input data projected on the Fisher subspace [64, 61, 52, 79, 43]. Nevertheless, although effective FLDA-based classification methods have been proposed, their performance is affected by some drawbacks that are shortly reported in the following.
At first, batch classification methods based on FLDA cannot efficiently deal with datasets of high dimensionality, for their high computational complexity.
Secondly, when the ratio between the number of training points and the dimensionality of the input space is close to 1 the classification performance drastically decreases. This is due to the fact that, under this setting, the sample covariance matrix is not a consistent estimator of the real covariance matrix [74].
We further note that when the number of training samples is smaller than the dimensionality of the samples, the within-class scatter matrix, $\boldsymbol{\Sigma}_W$, may become singular; in this case the execution of FLDA may encounter computational difficulty due to the need of computing the inverse of $\boldsymbol{\Sigma}_W$. Although many modified FLDA techniques have been proposed to overcome this problem (called the *small sample size problem*), it is still open.
Besides, as reported by some authors [100, 101], for a classification task with unbalanced classes the performance of FLDA based classifiers decrease.
Finally, since classifiers that exploit FLDA are linear techniques, they cannot cope with non linearly separable classes.

3.3 The Fisher's Criterion and Isotropy

In this section we report the Lemma proved by Brubaker and Vempala in [10] which is the basis of the IPCAC algorithm. The Lemma is the following, for

simplicity the within-class covariance matrix for the class c ($\mathbf{\Sigma}_{W_c}$) is denoted as $\mathbf{\Sigma}_c$:

Lemma 1. *Suppose* $\{w_c, \boldsymbol{\mu}_c, \mathbf{\Sigma}_c\}_{c=1}^C$ *defines an isotropic mixture in* $\Re^D$. *Let* $\lambda_1 \geq \cdots \geq \lambda_D$ *be the eigenvalues of the matrix* $\mathbf{\Sigma} = \sum_{c=1}^C w_c \mathbf{\Sigma_c}$ *and let* $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_D$ *be the corresponding eigenvectors. If the dimension of the span of the means of the components is* $C-1$, *then the Fisher subspace*

$$F = \mathbf{Span}\left\langle \boldsymbol{v}_{D-C+1}, \cdots, \boldsymbol{v}_D \right\rangle = \mathbf{Span}\left\langle \boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_C \right\rangle. \tag{3.10}$$

To prove this Lemma it is useful to recall that:

Fact 1. *Let* $\lambda_1 \geq \cdots \geq \lambda_D$ *be the eigenvalues for a* $D \times D$ *symmetric positive definite matrix* $\boldsymbol{Z}$ *and let* $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_D$ *be the corresponding eigenvectors. Then*

$$\lambda_D + \cdots + \lambda_{D-C+1} = \min_{\boldsymbol{\mathcal{S}}: dim(\boldsymbol{\mathcal{S}})=C} \sum_{j=1}^{C} \boldsymbol{p}_j^T \boldsymbol{Z} \boldsymbol{p}_j, \tag{3.11}$$

where $\{\boldsymbol{p}_j\}$ *is any orthonormal basis for the subspace* $\boldsymbol{\mathcal{S}}$. *If* $\lambda_{D-C} > \lambda_{D-C+1}$, *then* $\mathbf{Span}\left\langle \boldsymbol{v}_D, \cdots, \boldsymbol{v}_{D-C+1} \right\rangle$ *is the unique minimizing subspace.*

Proof of Lemma 1. By definition, for an isotropic distribution, the Fisher subspace minimizes

$$J(\boldsymbol{\mathcal{S}}) = \frac{\mathbb{E}_c\left[\mathbb{E}_i\left[\|proj_{\boldsymbol{\mathcal{S}}}(\boldsymbol{p}_{c,i} - \boldsymbol{\mu}_c)\|^2\right]\right]}{1} = \sum_{j=1}^{C-1} \boldsymbol{p}_j^T \mathbf{\Sigma} \boldsymbol{p}_j \tag{3.12}$$

where $\boldsymbol{p}_j$ is an orthonormal basis for $\boldsymbol{\mathcal{S}}$.

By Fact 1 the span of the smallest $C-1$ eigenvectors of the matrix $\mathbf{\Sigma}$, i.e. $\boldsymbol{v}_{D-C+2}, \cdots, \boldsymbol{v}_D$ is one of the minimizing subspaces. Being the distribution isotropic we have:

$$\mathbf{\Sigma} = \boldsymbol{I} - \sum_{c=1}^{C} \boldsymbol{w}_c \boldsymbol{\mu}_c \boldsymbol{\mu}_c^T, \tag{3.13}$$

and the vectors $\boldsymbol{v}_{D-C+2}, \cdots, \boldsymbol{v}_D$ become the largest eigenvectors of $\sum_{c=1}^C \boldsymbol{w}_c \boldsymbol{\mu}_c \boldsymbol{\mu}_c^T$. Moreover, since $\mathbf{Span}\left\langle \boldsymbol{v}_{D-C+2}, \cdots, \boldsymbol{v}_D \right\rangle \subseteq \mathbf{Span}\left\langle \mu_1, \cdots, \mu_C \right\rangle$, and

$$dim(\mathbf{Span}\left\langle \boldsymbol{v}_{D-C+2}, \cdots, \boldsymbol{v}_D \right\rangle) = C - 1 = dim(\mathbf{Span}\left\langle \boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_C \right\rangle),$$

it follows that the two subspaces are equal. Since $\boldsymbol{v}_{D-C+1}$ must be orthogonal to the other eigenvectors, it follows that $\lambda_{D-C+1} = 1 > \lambda_{D-C+2}$; therefore, $\mathbf{Span}\left\langle \boldsymbol{v}_{D-C+2}, \cdots, \boldsymbol{v}_D \right\rangle = \mathbf{Span}\left\langle \boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_C \right\rangle$ is the unique minimizing subspace.

3.4 IPCA-based Classifier

In Section 3.3 it is proved that, given a mixture of Gaussian distribution with mean $\boldsymbol{\mu} = \mathbf{0}$, and covariance matrix $\boldsymbol{\Sigma} = \sigma \boldsymbol{I}$ (where $\boldsymbol{I}$ is the identity matrix and σ the standard deviation), and given a set of clustered points $\boldsymbol{\mathcal{P}}$ sampled from it, then the subspace spanned by the cluster means approximates $\boldsymbol{F}\boldsymbol{s}_{\boldsymbol{\mathcal{P}}}$.

It follows that, in the two-class classification problem, $\boldsymbol{F}\boldsymbol{s}_{\boldsymbol{\mathcal{P}}}$ is one-dimensional and it is represented with a unit vector $\boldsymbol{F}$ computed as follows:

$$\boldsymbol{F} = \frac{\boldsymbol{\mu}_A - \boldsymbol{\mu}_B}{\|\boldsymbol{\mu}_A - \boldsymbol{\mu}_B\|} \tag{3.14}$$

where $\boldsymbol{\mu}_A \in \Re^D$ and $\boldsymbol{\mu}_B \in \Re^D$ are the means of the training points belonging to the classes A and B respectively.

In this case, classification could be achieved by computing $\boldsymbol{F}$, and thresholding the value of the projection of an unknown test point $\boldsymbol{p}$ on $\boldsymbol{F}$ as follows: $proj_{\boldsymbol{F}}(\boldsymbol{p}) = \boldsymbol{F} \cdot \boldsymbol{p}$, where $\cdot$ is the dot product operator.

Nevertheless, the probability distribution related to several classification tasks is not mean-centered, and its random variables are often correlated. For this reason, we preprocess the data by a linear (whitening) transformation[1]. Considering the set of N training points $\hat{\boldsymbol{\mathcal{P}}} = \bigcup_{c=1}^{C} \boldsymbol{\mathcal{P}}_c = \{\boldsymbol{p}_i\}_{i=1}^{N}$, the whitening matrix $\boldsymbol{W}$ and the overall mean $\tilde{\boldsymbol{\mu}}$ are estimated as described in Section 2.2.5:

1. estimate the expectation $\tilde{\boldsymbol{\mu}} = N^{-1} \sum_i \boldsymbol{p}_i$, and the sample covariance matrix $\tilde{\boldsymbol{\Sigma}} = N^{-1} \sum_i (\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})(\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})^T$;

2. estimate the principal components through the covariance matrix eigen-decomposition $\boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T = \tilde{\boldsymbol{\Sigma}}$;

3. estimate the whitening matrix as

$$\boldsymbol{W} = \boldsymbol{X}\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{X}^T \tag{3.15}$$

   Note that $\boldsymbol{\Lambda}^{-\frac{1}{2}}$ can be computed by substituting the non-zero diagonal elements λ_i of $\boldsymbol{\Lambda}$ with the values $\lambda_i^{-\frac{1}{2}}$.

[1] We call whitened data a set of points extracted from a multivariate probability distribution with $\boldsymbol{\mu} = \mathbf{0}$, and $\boldsymbol{\Sigma} = \boldsymbol{I}$.

The whitened training points, calculated through $\tilde{\boldsymbol{\mu}}$ and $\boldsymbol{W}$, are employed to compute the class means $\boldsymbol{\mu}_A$ and $\boldsymbol{\mu}_B$; $\boldsymbol{F}$ is then computed by means of Equation (3.14). Therefore, given a new point $\boldsymbol{p}$, it is projected on $\boldsymbol{F}$ as follows:

$$proj_{\boldsymbol{F}}(\boldsymbol{p}) = \boldsymbol{F}^T\boldsymbol{W}(\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) = \boldsymbol{w}^T(\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) \tag{3.16}$$

where $\boldsymbol{w} = \boldsymbol{W}^T\boldsymbol{F}$ is the vector used to simultaneously perform the data whitening and projection.

To classify the new point $\boldsymbol{p}$, a threshold γ must be determined, so that if $\boldsymbol{w} \cdot (\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) > \gamma$ than $\boldsymbol{p}$ is classified as belonging to class A, otherwise it is considered as belonging to class B. To estimate the best threshold value γ, we maximize the number of correctly classified points, that is:

$$\gamma = \left\langle \underset{\{\bar{\gamma}\} \subseteq \{\boldsymbol{w} \cdot (\boldsymbol{p}_i - \tilde{\boldsymbol{\mu}})\}}{\operatorname{argmax}} Score(\bar{\gamma}) \right\rangle \tag{3.17}$$

where the function $Score(\bar{\gamma})$ computes the number of correctly classified training points when $\bar{\gamma}$ is used as threshold, the argmax operator returns the set of thresholds $\{\bar{\gamma}\}$ leading to the maximum value, and $\langle \cdot \rangle$ is the mean operator.

The thresholding technique just described performs well in case of balanced training datasets; nevertheless, on unbalanced training data its performance decreases. To overcome this problem, we defined a new method for the computation of the thresholding value γ. More precisely, we consider the two sets $\bar{\gamma}_A$ and $\bar{\gamma}_B$ defined as:

$$\{\bar{\gamma}_A\} \equiv \{\boldsymbol{w}^T(\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) | \boldsymbol{p} \in \boldsymbol{\mathcal{P}}_A\} \tag{3.18}$$

$$\{\bar{\gamma}_B\} \equiv \{\boldsymbol{w}^T(\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) | \boldsymbol{p} \in \boldsymbol{\mathcal{P}}_B\} \tag{3.19}$$

where $\boldsymbol{\mathcal{P}}_A$ and $\boldsymbol{\mathcal{P}}_B$ are the training points belonging to class A and B respectively. We then compute the following maximal scores:

$$\gamma_A = \underset{\bar{\gamma} \in \{\bar{\gamma}_A\}}{\operatorname{argmax}} Score(\bar{\gamma}) \tag{3.20}$$

$$\gamma_B = \underset{\bar{\gamma} \in \{\bar{\gamma}_B\}}{\operatorname{argmax}} Score(\bar{\gamma}) \tag{3.21}$$

Using these values, the following threshold is selected:

$$\gamma = \frac{1}{2}\langle\bar{\gamma} \in \{\bar{\gamma}_A\} | Score(\bar{\gamma}) > k \cdot \gamma_A\rangle + \frac{1}{2}\langle\bar{\gamma} \in \{\bar{\gamma}_B\} | Score(\bar{\gamma}) > k \cdot \gamma_B\rangle \quad (3.22)$$

where $k \in (0, 1)$ is a parameter to be experimentally tuned, according to the unbalancing ratio.

The thresholding value computed by Equation (3.22) compensates the effects of unbalanced data, and allows to maintain a balanced amount of information from the two classes.

This classifier is very simple, and it has a low computational cost both in the training and in the classification phase. Moreover, denoting with D the feature space dimensionality, the space complexity required to store an IPCA-based classifier is at most $2D + 1$. Indeed, D real values are needed to store the estimated expectation[2] $\tilde{\boldsymbol{\mu}}$ (that will be denoted with $\boldsymbol{\mu}$ in the following), D real values record the weight vector $\boldsymbol{w}$, and one value stores the threshold γ.

3.4.1 IPCAC Retaining Variance

Given the matrix $\boldsymbol{P} \in \Re^{D \times N}$, representing a training dataset $\boldsymbol{\mathcal{P}} = \boldsymbol{\mathcal{P}}_A \cup \boldsymbol{\mathcal{P}}_B, |\boldsymbol{\mathcal{P}}| = N = N_A + N_B$, let α be the ratio D/N; assuming that the population covariance matrix $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^* + \sigma^2\boldsymbol{I}$, where $\boldsymbol{\Sigma}^*$ has rank $k < D$ and $\sigma^2\boldsymbol{I}$ represents the contribution of a zero mean Gaussian noise affecting the data, calling $\sigma^2 = \lambda_1 = \ldots = \lambda_{D-k-1} < \ldots < \lambda_D$ the ordered eigenvalues of $\boldsymbol{\Sigma}$, and denoting with $\tilde{\lambda}_1 < \ldots < \tilde{\lambda}_D$ the ordered eigenvalues of sample covariance matrix $\tilde{\boldsymbol{\Sigma}}$, in [75] it is proved that only the portion of the spectrum of $\boldsymbol{\Sigma}$ above $\sigma^2 + \sqrt{\alpha}$ can be correctly estimated from the sample.
To deal with the above mentioned problem, Belhumeur et al. [3] proposed a two-stage PCA+FLDA technique, that applies a first PCA step followed by the application of FLDA algorithm.
IPCAC solves this problem in a more efficient way. To describe it, we recall that, to evaluate the whitening $D \times D$ matrix $\boldsymbol{W}$ (see Equation (3.15)),

[2] Note that considering the equivalent thresholding function $\boldsymbol{w} \cdot \boldsymbol{p} > \gamma + \boldsymbol{w} \cdot \tilde{\boldsymbol{\mu}} \triangleq \beta$, the space required to store $\tilde{\boldsymbol{\mu}}$ can be avoided.

IPCAC performs an eigen-decomposition to diagonalize the overall sample covariance matrix. Nevertheless, it is possible to perform dimensionality reduction by selecting only the largest $d < D$ eigenvalues $(\tilde{\lambda}_1, \cdots, \tilde{\lambda}_d)$, and the corresponding eigenvectors, that retain a fixed percentage of variance $perc = \frac{\sum_{j=1}^{d} \tilde{\lambda}_j}{\sum_{j=1}^{N} \tilde{\lambda}_j}$; this allows to compute a matrix $\boldsymbol{W}_d \in \Re^{d \times D}$ that performs the whitening process on a d-dimensional subspace. After this step the Fisher subspace, spanned by the projection vector $\boldsymbol{F}_d$, is computed in the whitened d dimensional space. Therefore, given a new point $\boldsymbol{p} \in \Re^D$, it is projected on $\boldsymbol{F}_d$ as follows:

$$proj_{\boldsymbol{F}_d}(\boldsymbol{p}) = \boldsymbol{F}_d^T \boldsymbol{W}_d(\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) = \boldsymbol{w}_d^T(\boldsymbol{p} - \tilde{\boldsymbol{\mu}}) \tag{3.23}$$

where $\boldsymbol{w}_d = \boldsymbol{W}_d^T \boldsymbol{F}_d$ is the vector used to simultaneously perform the data whitening and projection in the d-dimensional subspace.

Our approach allows to achieve the same goal of the two-stage technique proposed by Belhumeur in a single step, thus reducing the time complexity of the algorithm.

3.5 Experimental Results on Syntethic Data

To assess the performance of the proposed techniques, we performed some tests on synthetic data comparing IPCAC, IPCAC retaining variance ($\texttt{IPCAC}_{var}$ with $perc = 0.99$), FLDA, and PCA+FLDA [3] (the PCA retained the 99% of the variance).

In order to produce synthetic datasets, we employed a generator that was built to draw points from two randomly selected non-isotropic Multivariate Gaussian Distributions (MGDs). To this aim, the mean vectors and the covariance matrices of the MGDs have been randomly choosen.

For each MGD, the mean vector coordinates were randomly selected in the open interval $(0, 1)$ according to the uniform distribution, while the covariance matrix was obtained by randomly generating both the positive eigenvalues, and the orthonormal eigenvectors. More precisely, the eigenvalues were generated by employing the random generator for the uniform distribution, while the eigenvectors were drawn from the normal distribution, and they were orthogonalized through the Gram-Schmidt orthogonalization process.

By employing this generator we produced 2 datasets, each one composed by two classes with 1000 points per class in $\Re^{500}$. The second dataset was created strongly reducing the distance between the class means, that is by multiplying the two means for a scale factor equal to 0.05.

We adopted K-fold cross validation ($K = 10$) to achieve an unbiased evaluation of the compared algorithms. To obtain a statistically significant evaluation, for each classifier we estimated the classification accuracy for all the K folds, and we executed R runs ($R = 50$) of K-folding permuting the data, thus obtaining KR accuracies to be averaged. All the classifiers are implemented in MATLAB.

The results obtained are shown in Table 3.1. From this table it can be

Classifier	Synth 1	Synth 2
IPCAC$_{var}$	98.50% ($\pm$0.0068)	83.28% ($\pm$0.0269)
IPCAC	81.28% ($\pm$0.0296)	68.01% ($\pm$0.0353)
FLDA	81.25% ($\pm$0.0289)	66.89% ($\pm$0.0332)
PCA+FLDA	96.70% ($\pm$0.0038)	80.91% ($\pm$0.0220)

Table 3.1: Experimental results (accuracy%, ($\pm$std)) obtained.

noticed that IPCAC and FLDA have the same behavior achieving comparable results. Applying the PCA step we obtained interesting improvements, as can be seen by observing the IPCAC$_{var}$ and PCA+FLDA results; note that IPCAC$_{var}$ achieves the best results.

3.6 IPCAC Model Merging

One of the benefits of using IPCAC, is the simplicity of combining several classifiers into a single one that has three main advantages: it is more accurate than a single IPCAC; it is simply upgradeable with dynamically supplied training data; it is able to cope with large training sets, since each classifier can be trained on a subset of the whole dataset.

To perform the IPCAC model merging (MM-IPCAC), the following informations must be stored for each trained classifier: $\{\boldsymbol{F}, \boldsymbol{W}, \boldsymbol{w}, \gamma, \boldsymbol{\mu}, \boldsymbol{\mu}_A, \boldsymbol{\mu}_B, N_A, N_B\}$, where N_A and N_B are the numbers of training feature vectors used in each class. We call $\{\boldsymbol{\mathcal{M}}_m\}_{m=1}^{M}$ the classification models to be merged, and

$\{\boldsymbol{F}_m, \boldsymbol{W}_m, \boldsymbol{w}_m, \cdots\}$ their components. Moreover, we consider the following quantities:

$$N_m = N_{Am} + N_{Bm}, \qquad N = \sum_m N_m$$
$$N_A = \sum_m N_{Am}, \qquad N_B = \sum_m N_{Bm}$$

At first, the new (merged) mean vector $\boldsymbol{\mu}$, and the merged whitening matrix $\boldsymbol{W}$, must be estimated for the whitening operation. More precisely, to merge the whitening matrices $\boldsymbol{W}_m$, the corresponding covariance matrices must be calculated. To this aim, we employ the eigen-decomposition $\boldsymbol{W}_m = \boldsymbol{X}_m \hat{\boldsymbol{\Lambda}}_m \boldsymbol{X}_m^T$, and we compute $\boldsymbol{\Sigma}_m = \boldsymbol{X}_m \hat{\boldsymbol{\Lambda}}_m^{-2} \boldsymbol{X}_m^T$. The merged mean vector $\boldsymbol{\mu}$, and the merged covariance matrix $\boldsymbol{\Sigma}$, are then estimated by means of the following generalizations of the results reported in [45]:

$$\begin{aligned}
\boldsymbol{\mu} &= N^{-1} \sum_{m=1}^{M} \boldsymbol{\mu}_m N_m \\
\boldsymbol{\Sigma} &= N^{-1} \sum_{m=1}^{M} \boldsymbol{\Sigma}_m N_m \\
&\quad + N^{-2} \sum_{m=1}^{M} \sum_{l=m+1}^{M} N_m N_l (\boldsymbol{\mu}_m - \boldsymbol{\mu}_l)(\boldsymbol{\mu}_m - \boldsymbol{\mu}_l)^T
\end{aligned}$$

To compute the merged whitening matrix $\boldsymbol{W}$ starting from $\boldsymbol{\Sigma}$, we use the eigen-decomposition $\boldsymbol{\Sigma} = \boldsymbol{X} \boldsymbol{\Lambda} \boldsymbol{X}^T$, and we compute $\boldsymbol{W} = \boldsymbol{X} \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{X}^T$.

To estimate $\boldsymbol{\mu}_A$ and $\boldsymbol{\mu}_B$ we recall that the means $\boldsymbol{\mu}_{Am}$ and $\boldsymbol{\mu}_{Bm}$, $m \in \{1, \ldots, M\}$, are determined on the whitened training points; therefore the merge operation requires the inversion of the whitening process. Being the whitening matrices $\boldsymbol{W}_m$ possibly singular, we use their pseudo-inverses denoted by $\boldsymbol{W}_m^\dagger$, and we merge the cluster means as follows:

$$\begin{aligned}
\boldsymbol{\mu}_A &= N^{-1} \boldsymbol{W} \sum_m \boldsymbol{W}_m^\dagger \boldsymbol{\mu}_{Am} N_{Am} \\
\boldsymbol{\mu}_B &= N^{-1} \boldsymbol{W} \sum_m \boldsymbol{W}_m^\dagger \boldsymbol{\mu}_{Bm} N_{Bm}
\end{aligned}$$

where N_{Am} and N_{Bm} are the numbers of training data points used to estimate $\boldsymbol{\mu}_{Am}$ and $\boldsymbol{\mu}_{Bm}$.

Given the quantities computed above, the new vector $\boldsymbol{F}$, and the weight vector $\boldsymbol{w}$, are computed as before:

$$\boldsymbol{F} = \frac{\boldsymbol{\mu}_A - \boldsymbol{\mu}_B}{\|\boldsymbol{\mu}_A - \boldsymbol{\mu}_B\|}; \qquad \boldsymbol{w} = \boldsymbol{W}^T \boldsymbol{F}$$

The last model part to be merged is the thresholding value γ; to get it, for each classifier m we compute the point $\hat{\boldsymbol{\gamma}}_m$ on $\boldsymbol{F}\boldsymbol{s}_{\boldsymbol{\mathcal{P}}}$ such that its projection on $\boldsymbol{F}_m$ generates exactly the thresholding value γ_m. We get:

$$\begin{aligned} \gamma_m = \boldsymbol{F}_m^T \boldsymbol{W}_m(\hat{\boldsymbol{\gamma}}_m - \boldsymbol{\mu}_m) \quad &\Rightarrow \quad \gamma_m \boldsymbol{F}_m = \boldsymbol{W}_m(\hat{\boldsymbol{\gamma}}_m - \boldsymbol{\mu}_m) \\ &\Rightarrow \quad \hat{\boldsymbol{\gamma}}_m = \gamma_m \boldsymbol{W}_m^{\dagger} \boldsymbol{F}_m + \boldsymbol{\mu}_m \end{aligned}$$

The merged threshold γ is then computed by projecting the average of the $\hat{\boldsymbol{\gamma}}_m$ on $\boldsymbol{F}$, that is:

$$\begin{aligned} \hat{\boldsymbol{\gamma}} &= N^{-1} \sum_m \hat{\boldsymbol{\gamma}}_m N_m \\ \gamma &= \boldsymbol{F} \cdot \boldsymbol{W}(\hat{\boldsymbol{\gamma}} - \boldsymbol{\mu}) = \boldsymbol{w}^T(\hat{\boldsymbol{\gamma}} - \boldsymbol{\mu}) \end{aligned}$$

Note that, the obtained classification model maintains the same space and time complexity of the original ones.

3.7 A Real Application: the Spam Classification

The proposed classification methods have been tested on email classification to recognize spam emails from legitimate emails. In this section we describe the framework implemented for our tests, the experimental setting, and the obtained results.

Our spam filter is a classification system based on email text semantic analysis. The following tasks are performed to achieve the message classification:

Sparse vectors construction: each email is processed to be represented as an high dimensional, and sparse feature vector. More precisely, the words contained in each email are extracted and reduced to the same form by means of standard stemming algorithms [62]; moreover, the obtained set of words is filtered by removing unknown terms (i.e. terms

that are not listed in a predefined 'dictionary'[3]). The sparse feature vector is then defined by the frequencies of the remaining words in the processed email.

Sparse to dense projection: in order to generate an easily manipulable representation, and to extract semantic informations, the sparse feature vectors are projected on a lower dimensional space. To this aim, the Term Frequency-Inverse Document Frequency coefficients (`TF-IDF`, [5]), and the Latent Semantic Analysis technique (`LSA`, [57]), are applied to the sparse feature vectors in the training set. These algorithms generate a sparse to dense projection matrix $\boldsymbol{SD}$ that is used to compute all the dense feature vectors processed by the classifiers. We must highlight that sparse to dense projection matrices computed on different training sets allow to compute dense feature vectors of different dimensionality and semantic.

Classification: classification is performed by applying the chosen classifier to the obtained dense feature vectors.

3.7.1 Experiments

In order to show the effectiveness of our algorithms, we have compared their results to those obtained by well-known methods described in the literature, more precisely `SVM` and `KNN`[4].

The tests have been performed on two standard email sets, that are: 12000 messages randomly extracted from the TREC 2005 corpus [24] (6000 ham, and 6000 spam), and 3600 messages randomly selected from the SpamAssassin corpus (1800 ham, and 1800 spam) [1]. Each database has been randomly halved, thus obtaining $Dataset1$ and $Dataset2$, and the following two experiments have been carried out:

Experiment 1: in this experiment we tested the $\texttt{IPCAC}_{var}$[5], `SVM`, and `KNN`

[3]The employed dictionary contains approximately 87000 words and acronyms extracted from different sources.

[4]For each classifier, we determined the best configuration parameters by executing a tuning phase on a smaller data-set, that was automatically generated through random message selection and K-folding.

[5]We retain the 99% of the variance.

classifiers by performing 4-fold cross validation on $Dataset1$ and we averaged the evaluation measures.

Experiment 2: this experiment has been performed both to test the robustness of all the classifiers with respect to input vectors generated through different sparse to dense projection matrices, and to compare the MM-IPCAC classifier with the other techniques. To this aim, the following steps were performed:

- a sparse to dense projection matrix $\hat{\boldsymbol{SD}}$ was calculated using half of $Dataset1$;
- the matrix $\hat{\boldsymbol{SD}}$ was employed to process the second part of $Dataset1$;
- the obtained set of dense vectors was randomly split into four overlapped subsets that were used to train four $\texttt{IPCAC}_{var}$, four SVMs, and four KNN models;
- the MM-IPCAC technique was applied to merge the four $\texttt{IPCAC}_{var}$ classifiers;
- all the trained classifiers were then tested on the feature vectors obtained by processing $Dataset2$ with $\hat{\boldsymbol{SD}}$. The performance of the four $\texttt{IPCAC}_{var}$, the four SVMs, and the four KNN classifiers were averaged to obtain their final result.

All the described experiments were executed by using a Python implementation for the framework and for all the tested algorithms.[6].

3.7.2 Obtained Results

We executed our tests using as evaluation measures the *Accuracy*, *Recall* (or *Sensitivity*), and *Precision* (see Appendix A), where we considered as positive examples the spam messages.

The achieved performances have been reported in Table 3.2 and Table 3.3. From this results the reader can note that:

[6]For the SVM classifier we used a Python wrapper to the library "libSVM" [12]. For the KNN algorithm we used the implementation of Biopython [22]

Experiment$_{\#}$	*Classifier*	*Accuracy (std)*	*Precision*	*Recall*
Exp_1	KNN	95.499 (0.759)	95.922	95.028
	IPCAC$_{var}$	96.817 (0.279)	96.250	97.430
	SVM	**97.116 (0.213)**	**96.234**	**98.057**
Exp_2	KNN	95.449	95.113	95.927
	IPCAC$_{var}$	95.05	93.901	96.374
	SVM	97.001	95.958	98.133
	MM-IPCAC	**98.350**	**97.387**	**99.367**

Table 3.2: Experimental results on the emails belonging to TREC corpus.

Experiment$_{\#}$	*Classifier*	*Accuracy (std)*	*Precision*	*Recall*
Exp_1	KNN	96.278 (0.379)	96.773	95.777
	IPCAC$_{var}$	97.444 (1.002)	97.608	97.333
	SVM	**98.444 (0.602)**	**98.148**	**98.778**
Exp_2	KNN	91.222	96.932	85.111
	IPCAC$_{var}$	90.167	93.280	86.667
	SVM	93.611	96.444	90.556
	MM-IPCAC	***98.222***	***98.329***	***98.111***

Table 3.3: Experimental results on the emails belonging to SpamAssassin corpus.

Experiment 1: on both corpuses the results achieved by IPCAC$_{var}$, and SVM (employing a gaussian non-linear kernel properly tuned) are comparable, while KNN achieves lower performances.

Experiment 2: on both corpuses the results show that our MM-IPCAC technique is promising, since it outperforms all the other algorithms.

It is important to highlight that all the classifiers, except MM-IPCAC, obtain low accuracy when *Experiment*2 is run on the SpamAssassin corpus; this is due to the fact that $\hat{\boldsymbol{SD}}$ is computed exploiting a subset of the *Dataset*1 having a low cardinality; this produces a not accurate result affected by discriminative information loss, that decreases the classification performance.

These experiments confirm the effectiveness of the proposed algorithms.

3.8 Conclusion

In this chapter a linear classification algorithm, called IPCAC, is described in detail; this classifier deals with data drawn from a mixture of two gaussians. Furthermore, we present an improved version of this method called $\texttt{IPCAC}_{var}$, that performs an implicit dimensionality reduction to garantee a better performance. IPCAC and $\texttt{IPCAC}_{var}$ have been tested on synthetic and real data, achieving promising results.

Finally, to cope with large datasets or with dynamically supplied data, a Model Merging technique has presented to merge different IPCAC, or $\texttt{IPCAC}_{var}$, trained models. Tests on spam classification, and the comparison with SVM, KNN have demonstrated the effectiveness of this method.

Chapter 4

O-IPCAC: an Online Classifier based on IPCAC

In practical application the cardinality of the training dataset (N) is often lower than the space dimensionality (D), or the ratio between N and D is approximately 1. Both these conditions might cause hard problems to classifiers exploiting the sample covariance matrix during their computations. Firstly, when $N < D$ the *small sample size problem* might occur, causing the covariance matrix to be singular and hereby raising computational problems during the learning and classification tasks. On the other side, when $N \approx D$ the sample covariance matrix is an inconsistent estimator of the population covariance matrix, as proved by Johnstone et al. in [55], thus affecting the classification performance. It has to be further noted that, when either the training set cardinality or the input space dimensionality are high, several classifiers cannot be applied for their high computational costs.

Despite the effectiveness of IPCAC, its performance is highly affected under the above mentioned circumstances; for this reason, in this chapter we propose its improvement to address these problems, that will be referred as O-IPCAC (Online IPCAC). More precisely, in O-IPCAC two main improvements have been defined: firstly, problems due to the high-dimensional data are tackled by replacing the data whitening with a process that whitens the data in a linear subspace $\pi_d = \mathbf{Span}\left\langle \boldsymbol{v}_1, \ldots, \boldsymbol{v}_d \right\rangle, d \ll D$, while maintaining unaltered the information related to the orthogonal subspace $(\pi_d)^{\perp} = \mathbf{Span}\left\langle \boldsymbol{v}_{d+1}, \ldots, \boldsymbol{v}_D \right\rangle$; secondly, to cope with huge amount of training points and with data dynamically supplied the classification algorithm has been

designed to perform online/incremental training. As result, we obtain a classifier that can deal with problems where $N \approx D$, and both these values are high.

We demonstrate the efficacy of our algorithm by employing it for the classification of both synthetic and real datasets, and by comparing the computed results with those achieved by other well-known methods.

The chapter is organized as follows: in Section 4.1 we briefly overview related works that deal with the small sample size problem; in Section 4.2 the method to perform Incremental Singular Value Decomposition is summarized; in Section 4.3 we present our method, called Online-IPCAC, and its adaptive version that deals with problems where the probability distribution underlying the data might change with time; in Section 4.4 we show the results obtained by our method on different datasets and we compare it with well-known batch and online classifiers; in Section 4.5 we present the results achieved by O-IPCAC when it is applied for EEG data classification.

4.1 Related works

At the state of the art, many techniques based on FLDA have been proposed to deal with high dimensional data and with the *small sample size problem* (for a detailed review see [13]). We recall that, as mentioned in Section 3.2 this problem occurs when the number of training samples is lower than the space dimensionality, hereby producing a singular within-class scatter matrix $\mathbf{\Sigma}_W$. This makes it difficult to compute the set of projection vectors spanning the Fisher subspace. In this section we briefly overview the most important techniques that try to cope with the mentioned problem.

The first authors that tried to tackle the small sample size problem, applied linear algebra techniques to solve the numerical problems due to the singularity of the sample within-class covariance matrix. As an example, in [92] the authors employ the pseudo inverse of the scatter matrix to compute the Fisher subspace; alternatively, some researches [50, 103] recommend the addition of a small perturbation to the within-class scatter matrix so that it becomes nonsingular.

Although interesting results have been reported in the above mentioned works, the most promising and recent techniques employ a subspace approach. Among them, one of the most notable methods is a two-stage `PCA+FLDA` [91, 3] that performs a first step to evaluate the Principal Components used to project the point into a subspace where the sample within-class scatter matrix is not degenerate; this allows to subsequently apply the `FLDA` algorithm without any problem. The drawback of this method is due to the fact that dimensionality reduction might discard important discriminative information, thus decreasing the classification performance. As an example, consider two classes with the shape of two parallel "pancakes" in $\Re^D$, i.e. two Gaussians that are spherical in $D-1$ directions and narrow in the last direction (see Figure 4.1), so that an hyperplane orthogonal to the last direction separates the two. Under these circumstances, dimensionality reduction through `PCA` would keep the directions of maximum elongation while discarding the last direction. As a consequence, the result of the data projection process would be a mixture of two completely overlapped Gaussian distributions.

Chen et al. [13] proposed a different approach, which exploits the modified Fisher's criterion presented by Liu in [60]. We recall that this Fisher's criterion (see Equation (3.9)) employs the total sample covariance matrix ($\mathbf{\Sigma}_{Tot} = \mathbf{\Sigma}_{Bet} + \mathbf{\Sigma}_W$) as the divisor of the original Fisher's criterion, instead of merely using the within-class scatter matrix ($\mathbf{\Sigma}_W$).

In cases of nonsingular $\mathbf{\Sigma}_W$, Chen et al. compute the optimal discriminant projection vectors according to [60], that is by selecting the eigenvectors corresponding to the set of the largest eigenvalues of the matrix $\mathbf{\Sigma}_{Tot}^{-1}\mathbf{\Sigma}_{Bet}$. On the other side, when the small sample size problem occurs so that $\mathbf{\Sigma}_W$ (and hence $\mathbf{\Sigma}_{Tot}$) is singular, the authors compute the Fisher subspace on the points projected in the *null* space of $\mathbf{\Sigma}_W$. More precisely, if the rank of $\mathbf{\Sigma}_W$ is equal to r, the *null* space is the subspace $R_{null} \subset \Re^D$ such that

$$R_{null} = \mathbf{Span}\left\langle \alpha_i \in \Re^D | \mathbf{\Sigma}_W \alpha_i = 0, i = 1, \cdots, D-r \right\rangle$$

Note that the existence of this space is guaranteed by the fact that r is lower than the space dimensionality, D. Considering $\boldsymbol{Q} = [\alpha_1, \cdots, \alpha_{D-r}]$, then the transformation $\boldsymbol{Q}\boldsymbol{Q}^T$ can be used to transform all the vectors $\boldsymbol{x}$

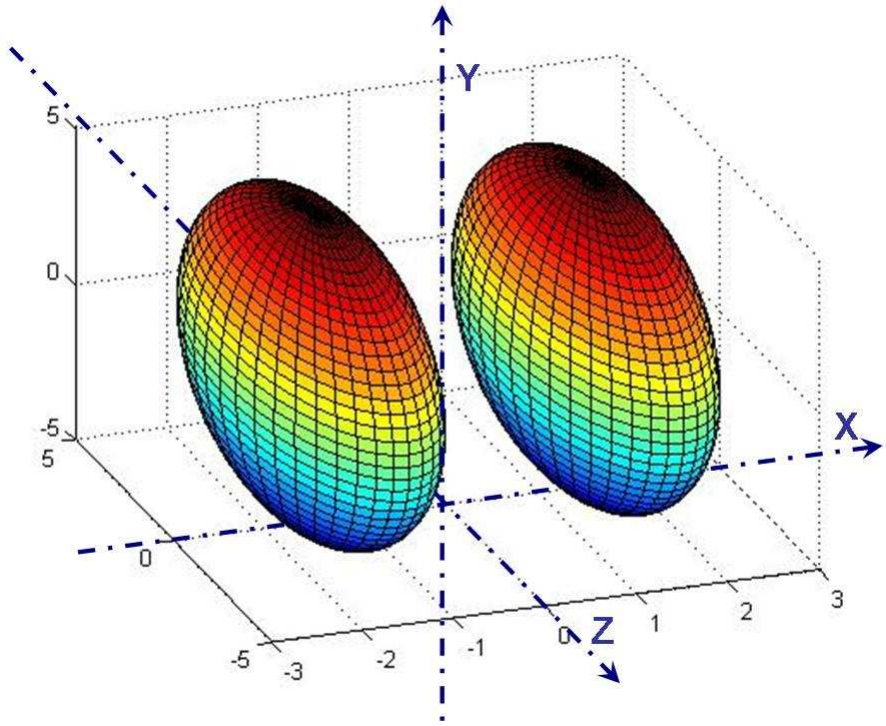


Figure 4.1: Two classes with the shape of two parallel "pancakes"; they are represented by two ellipsoids that are spherical along the Y and Z directions, and narrow along the X axis. Discarding the direction of the lowest variance, that is oriented along the X axis, the projected classes would be shaped as two completely overlapped circles on the Y-Z plane.

from $\Re^D$ into its subspace R_{null}. From the linear algebra it can be easily proved that for each $\boldsymbol{w} \in R_{null}$ then $\boldsymbol{w}^T \boldsymbol{\Sigma}_W \boldsymbol{w} = 0$, so that it would be $\boldsymbol{\Sigma}_{Tot} = \boldsymbol{\Sigma}_{Bet}$. In this case, the authors note that the modified Fisher's criterion becomes equal to 1, that is its maximum, for each $\boldsymbol{w} \in R_{null}$ such that $\boldsymbol{w}^T \boldsymbol{\Sigma}_{Bet} \boldsymbol{w} \neq 0$. However, a projection vector $\boldsymbol{w} \in R_{null}$ satisfying the above mentioned conditions cannot guarantee the maximum class separability, unless $\boldsymbol{w}^T \boldsymbol{\Sigma}_{Bet} \boldsymbol{w}$ is further maximized. Therefore, Chen et al. propose to choose, as projection vectors, those vectors in the `null` space that maximize the between-class scatter of the training samples transformed into R_{null}. This procedure is based on the authors' observation that the `null` space of $\boldsymbol{\Sigma}_W$ carries the most of the discriminative information. Intuitively, choosing projection vectors such that $\boldsymbol{w}^T \boldsymbol{\Sigma}_W \boldsymbol{w} = 0$ while $\boldsymbol{w}^T \boldsymbol{\Sigma}_{Bet} \boldsymbol{w} \neq 0$ means

that, when projected onto $\boldsymbol{w}$, the within-class separation is abolished while the between-class scatter is not. Obviously, this method has the effect of increasing the classification performance. Nevertheless, the main drawback of this technique is due to the fact that it does not consider any information outside the `null` space of $\boldsymbol{\Sigma}_W$; furthermore, it has some computational limitations in managing large sample covariance matrices.
To overcome these limitations, Hua Yu and Jie Yang [54] proposed a "unified" `PCA+FLDA`, called `D-FLDA`, which simultaneously performs the two steps. The key idea of this algorithm is to discard the `null` space of $\boldsymbol{\Sigma}_{Bet}$, which does not contain useful discriminative information, maintaining the information contained in the `null` space of $\boldsymbol{\Sigma}_W$ that, as proved by Chen, is the most discriminative part. To this aim, the authors proposed to proceed in the following order:

- diagonalize $\boldsymbol{\Sigma}_{Bet}$, that is find a matrix $\boldsymbol{V}$ such that $\boldsymbol{V}^T\boldsymbol{\Sigma}_{Bet}\boldsymbol{V} = \boldsymbol{\Lambda}$, $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix sorted in decreasing order;

- diagonalize $\boldsymbol{Y}^T\boldsymbol{\Sigma}_W\boldsymbol{Y}$, where $\boldsymbol{Y}$ is composed of the first columns of $\boldsymbol{V}$, and discard the largest eigenvalue/eigenvector pairs to keep the information in the `null` space of $\boldsymbol{\Sigma_W}$.

Although this approach reverses the traditional procedure order, when $\boldsymbol{\Sigma}_W$ is not singular it is equivalent to `PCA+FLDA` [3]; otherwise, when $\boldsymbol{\Sigma}_W$ is singular the reversal in order makes a drastic difference.
Lu et al. in [63] noted that the performance of the technique proposed by Hua Yu and Jie Yang deteriorates rapidly when the small sample size problem becomes severe, since the insufficient number of training examples makes it difficult to compute a reliable estimate of the null eigenvalues of the within-class covariance matrix, and hereby its `null` space. Indeed, in this case, both bias and variance affect the estimation of the within-class covariance matrix. Firstly, the $\boldsymbol{\Sigma}_W$ estimate produces biased estimated eigenvalues, so that the largest ones are biased high and the smallest ones are biased toward too low values. Secondly, the estimate of the `null` space of $\boldsymbol{\Sigma_W}$ can be highly unstable, giving rise to high variance. To solve this problem, the authors proposed a new `FLDA` methodology (`R-FLDA`, [63, 64])

based on a regularized Fisher's discriminant criterion, that is:

$$J(\boldsymbol{W}) = \frac{\boldsymbol{W}^T \boldsymbol{\Sigma}_{Bet} \boldsymbol{W}}{\nu(\boldsymbol{W}^T \boldsymbol{\Sigma}_{Bet} \boldsymbol{W}) + \boldsymbol{W}^T \boldsymbol{\Sigma}_W \boldsymbol{W}} \qquad 0 \leq \nu \leq 1$$

which is equivalent to both D-FLDA and FLDA under particular configurations of the regularization parameter ν. This method is inspired to the regularized quadratic discriminant analysis proposed by Friedman [36], where a regularization term is introduced to cope with highly ill-posed covariance matrices. Friedman demonstrated that the regularization successfully decreases the larger eigenvalues and increases the smaller ones, thereby counteracting the biasing. Moreover, this technique stabilizes the smallest eigenvalues. Indeed, experimental results presented in [63, 64] showed that the regularization technique inspired to Friedman's approach is more robust, against the small sample size problem, than the D-FLDA technique.

We note that, although promising results have been obtained by the above mentioned techniques, all of them do not consider the case where the number, N, of training examples is approximately equal to the space dimensionality ($N \approx D$). In such circumstances, since it might be $N > D$, the sample within-class covariance matrix is nonsingular but the sample covariance matrix is still not a consistent estimator of the population covariance matrix, as proved in [55]. Under this setting, the estimated null space might not contain useful information, thus causing a drastic performance decrease.

4.2 Incremental Singular Value Decomposition

In this section the method proposed by Brand [7] to incrementally perform the singular value decomposition (SVD) is described in detail since it is used in the rest of this chapter.

Given a real matrix $\boldsymbol{X} \in \Re^{D \times N}$ with rank d and economy SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T$, where $\boldsymbol{Q}$ in $\Re^{d \times d}$, Brand proposed [7] an identity for additive modifications of a SVD to reflect updates, shifts, downdates, and edits of the data matrix, $\boldsymbol{X}$. More precisely, considering two arbitrary matrices $\boldsymbol{A} \in \Re^{D \times c}$

and $\boldsymbol{B} \in \Re^{N \times c}$ of rank c, he showed how to express the SVD of the sum:

$$\boldsymbol{X} + \boldsymbol{A}\boldsymbol{B}^T = \begin{bmatrix} \boldsymbol{U} & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{V} & \boldsymbol{B}^T \end{bmatrix} \tag{4.1}$$

as modifications to $\boldsymbol{U}, \boldsymbol{Q}, \boldsymbol{V}$; in this section we briefly recall the method proposed by Brand.

Let $\boldsymbol{P}$ an orthogonal basis of the column-space of $\left(\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^T\right)\boldsymbol{A}$, and $\boldsymbol{Q}$ an orthogonal basis of the column-space of $\left(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^T\right)\boldsymbol{B}$; setting

$$\boldsymbol{R_A} = \boldsymbol{P}^T\left(\boldsymbol{I} - \boldsymbol{U}\boldsymbol{U}^T\right)\boldsymbol{A} \qquad \textit{and} \qquad \boldsymbol{R_B} = \boldsymbol{Q}^T\left(\boldsymbol{I} - \boldsymbol{V}\boldsymbol{V}^T\right)\boldsymbol{B}$$

we obtain:

$$\boldsymbol{X} + \boldsymbol{A}\boldsymbol{B}^T = \begin{bmatrix} \boldsymbol{U} & \boldsymbol{P} \end{bmatrix} \boldsymbol{K} \begin{bmatrix} \boldsymbol{V} & \boldsymbol{Q} \end{bmatrix}^T \tag{4.2}$$

where the matrix $\boldsymbol{K}$ is:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{U}^T\boldsymbol{A}^T \\ \boldsymbol{0} & \boldsymbol{R_A} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{V}^T\boldsymbol{B}^T \\ \boldsymbol{0} & \boldsymbol{R_B} \end{bmatrix}^T = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{U}^T\boldsymbol{A} \\ \boldsymbol{R_A} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}^T\boldsymbol{B} \\ \boldsymbol{R_B} \end{bmatrix}^T \tag{4.3}$$

which is usually small, highly structured, and sparse; if we diagonalize $\boldsymbol{K}$ as $\boldsymbol{U'}^T\boldsymbol{K}\boldsymbol{V'} = \boldsymbol{Q'}$ we obtain the rotations $\boldsymbol{U'}$ and $\boldsymbol{V'}$ of the subspaces $\begin{bmatrix} \boldsymbol{U} & \boldsymbol{P} \end{bmatrix}$ and $\begin{bmatrix} \boldsymbol{V} & \boldsymbol{Q} \end{bmatrix}^T$. Reformulating Equation (4.3), the following identity is obtained:

$$\boldsymbol{X} + \boldsymbol{A}\boldsymbol{B}^T = \left(\begin{bmatrix} \boldsymbol{U} & \boldsymbol{P} \end{bmatrix} \boldsymbol{U'}\right) \boldsymbol{Q'} \left(\begin{bmatrix} \boldsymbol{V} & \boldsymbol{Q} \end{bmatrix} \boldsymbol{V'}\right)^T \tag{4.4}$$

that is the desired SVD.

Given the vector $\boldsymbol{a} \in \Re^D$ and $\boldsymbol{b} \in \Re^N$, it can be useful to consider
$\boldsymbol{m} = \boldsymbol{U}^T\boldsymbol{a}$; $\boldsymbol{p} = \boldsymbol{a} - \boldsymbol{U}\boldsymbol{m}$; $R_a = \|\boldsymbol{p}\|$; $\boldsymbol{P} = R_a^{-1} \cdot \boldsymbol{p}$; and
$\boldsymbol{n} = \boldsymbol{V}^T\boldsymbol{b}$; $\boldsymbol{q} = \boldsymbol{b} - \boldsymbol{V}\boldsymbol{n}$; $R_b = \|\boldsymbol{q}\|$; $\boldsymbol{Q} = R_b^{-1} \cdot \boldsymbol{q}$.
Under this setting, the diagonalisation problem of Equation (4.3) is simplified to:

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{m} \\ \|\boldsymbol{p}\| \end{bmatrix} \begin{bmatrix} \boldsymbol{n} \\ \|\boldsymbol{q}\| \end{bmatrix}^T \tag{4.5}$$

	Before	After	$\boldsymbol{a}$	$\boldsymbol{b}^T$
Update	$\boldsymbol{U}\boldsymbol{Q}\left[\boldsymbol{V}^T \quad \boldsymbol{0}\right] = \left[\boldsymbol{X} \quad \boldsymbol{0}\right]$	$\boldsymbol{U}'\boldsymbol{Q}'\boldsymbol{V}'^T = \left[\boldsymbol{X} \quad \boldsymbol{c}\right]$	$\boldsymbol{c}$	$[0, \cdots, 0, 1]$
Downdate	$\boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T = \left[\boldsymbol{X} \quad \boldsymbol{c}\right]$	$\boldsymbol{U}'\boldsymbol{Q}'\boldsymbol{V}'^T = \boldsymbol{X}$	$-\boldsymbol{c}$	$[0, \cdots, 0, 1]$
Recenter	$\boldsymbol{U}\boldsymbol{Q}\boldsymbol{V}^T = \boldsymbol{X}$	$\boldsymbol{U}'\boldsymbol{Q}'\boldsymbol{V}'^T = \boldsymbol{X}\left(\boldsymbol{I} - \frac{1}{q}\boldsymbol{1}\boldsymbol{1}^T\right)$	$-\frac{1}{q}\boldsymbol{X}\boldsymbol{1}$	$\boldsymbol{1}^T = [1, \cdots, 1]$

Table 4.1: The operations expressed as rank-1 modifications of an SVD, where $\boldsymbol{c}$ is a vector in $\Re^p$ representing the modification element.

The method described in this section can be used for computing a thin SVD of streaming data in a single pass with linear time complexity. Indeed, considering a rank-d thin SVD of a $D \times N$ matrix, the necessary d-modifications can be computed in $O(dDN)$ time for $d \leq \sqrt{min(D, N)}$.
In Table 4.2 we summarize the updating, downdating, and recentering operations that are expressed as specializations of this scheme.

4.3 Online IPCAC

Given the matrix $\boldsymbol{P} \in \Re^{D \times N}$, representing a training dataset $\boldsymbol{\mathcal{P}} = \boldsymbol{\mathcal{P}}_A \cup \boldsymbol{\mathcal{P}}_B, |\boldsymbol{\mathcal{P}}| = N = N_A + N_B$, let α be the ratio D/N; when $\alpha \approx 1$ the performance of IPCAC (see Section 3.4) deteriorates dramatically since the sample covariance matrix $\boldsymbol{S}_N = \frac{1}{N-1}\boldsymbol{P}\boldsymbol{P}^T$ is not a consistent estimator of the population covariance matrix $\boldsymbol{\Sigma}$ [55]. More precisely, assuming that $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^* + \sigma^2\boldsymbol{I}$, where $\boldsymbol{\Sigma}^*$ has rank $k < D$ and $\sigma^2\boldsymbol{I}$ represents the contribution of a zero mean Gaussian noise affecting the data, calling $\sigma^2 = \lambda_1 = \ldots = \lambda_{D-k-1} < \ldots < \lambda_D$ the ordered eigenvalues of $\boldsymbol{\Sigma}$, and denoting with $l_1 < \ldots < l_D$ the ordered eigenvalues of $\boldsymbol{S}_N$, in [75] it is proved that only the portion of the spectrum of $\boldsymbol{\Sigma}$ above $\sigma^2 + \sqrt{\alpha}$ can be correctly estimated from the sample. Furthermore, when $\alpha \approx 1$ the estimates of the smallest eigenvalues l_i can be much smaller than the real ones, and the corresponding estimated eigenvectors are uncorrelated with the real ones. These results motivate our choice of improving IPCAC by obliging it to consider only the largest eigenvalues; this is obtained by substituting the whitening step by a "partial" whitening with respect to the first $d \leq D$ principal components of

$\boldsymbol{P}$, where d is a parameter to be set[1].

To estimate the linear transformation $\boldsymbol{W}$, which represents the partial whitening operator, we apply the Truncated Singular Value Decomposition[2] (TSVD, [46]), obtaining the low-rank factorization $\boldsymbol{P} \simeq \boldsymbol{U}_d \boldsymbol{Q}_d \boldsymbol{V}_d^T$. The d largest singular values on the diagonal of $\boldsymbol{Q}_d$, and the associated left singular vectors, are employed to project the points in $\boldsymbol{P}$ on the subspace $\boldsymbol{\mathcal{SP}}_d$ spanned by the columns of $\boldsymbol{U}_d$, and to perform the whitening, as follows:

$$\bar{\boldsymbol{P}}_{\boldsymbol{W}_d} = q_d \boldsymbol{Q}_d^{-1} \boldsymbol{P}_{\perp \boldsymbol{\mathcal{SP}}_d} = q_d \boldsymbol{Q}_d^{-1} \boldsymbol{U}_d^T \boldsymbol{P} = \boldsymbol{W}_d \boldsymbol{P} \tag{4.6}$$

where q_d is the smallest singular value of the points projected on $\boldsymbol{\mathcal{SP}}_d$. Note that, to obtain points whose covariance matrix best resembles a multiple of the identity, we have chosen to set the value of the d largest singular values to q_d instead of 1, thus avoiding the gap between the d-th and the $(d+1)$-th singular value.

The obtained matrix $\boldsymbol{W}_d$ projects and whitens the points in the linear subspace $\boldsymbol{\mathcal{SP}}_d$; however, dimensionality reduction might delete discriminative information, decreasing the classification performance as reported in [54] (see Section 4.1 and Figure 4.1).

To avoid this information loss, we add to the partially whitened data the residuals, $\boldsymbol{R}$, of the points in $\boldsymbol{P}$ with respect to their projections on $\boldsymbol{\mathcal{SP}}_d$:

$$\begin{aligned}
\boldsymbol{R} &= \boldsymbol{P} - \boldsymbol{U}_d \boldsymbol{P}_{\perp \boldsymbol{\mathcal{SP}}_d} = \boldsymbol{P} - \boldsymbol{U}_d \boldsymbol{U}_d^T \boldsymbol{P} \\
\bar{\boldsymbol{P}}_{\boldsymbol{W}_D} &= \boldsymbol{U}_d \bar{\boldsymbol{P}}_{\boldsymbol{W}_d} + \boldsymbol{R} \\
&= \boldsymbol{U}_d \boldsymbol{W}_d \boldsymbol{P} + \boldsymbol{P} - \boldsymbol{U}_d \boldsymbol{U}_d^T \boldsymbol{P} \\
&= \left(q_d \boldsymbol{U}_d \boldsymbol{Q}_d^{-1} \boldsymbol{U}_d^T + \boldsymbol{I} - \boldsymbol{U}_d \boldsymbol{U}_d^T \right) \boldsymbol{P} \\
&= \boldsymbol{W} \boldsymbol{P}
\end{aligned}$$

where $\boldsymbol{W} \in \Re^{D \times D}$ represents the linear transformation that whitens the data along the first d principal components, while keeping unaltered the information along the remaining components.

[1] We empirically chose $d = \min(\log_2^2 N, D)$ since our tests on synthetic data showed that the generalization capability of O-IPCAC remains approximately maximal by employing this value (see Section 4.3.2).

[2] Since the sample covariance matrix estimation is $\boldsymbol{\Sigma_N} = \frac{1}{N-1} \boldsymbol{P} \boldsymbol{P}^T$, the eigen-decomposition of $\boldsymbol{\Sigma_N} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^T$ is performed by calculating the SVD decomposition of $\boldsymbol{P}$, that is $\boldsymbol{P} = \boldsymbol{U} \boldsymbol{Q} \boldsymbol{V}^T$ (see Section 2.2.2); since $\boldsymbol{P} \boldsymbol{P}^T = \boldsymbol{U} \boldsymbol{Q}^2 \boldsymbol{U}^T$ and setting $\boldsymbol{\Lambda} = \frac{1}{N-1} \boldsymbol{Q}^2$ we obtain $\boldsymbol{\Sigma_N} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^T$.

The Fisher subspace is then estimated by exploiting the whitened class means, $\boldsymbol{\mu}_A$ and $\boldsymbol{\mu}_B$, obtained by applying the partial whitening transform to the class means in the original space $\hat{\boldsymbol{\mu}}_A$ and $\hat{\boldsymbol{\mu}}_B$, as follows:

$$\begin{aligned}\boldsymbol{\mu}_A &= \boldsymbol{W}\hat{\boldsymbol{\mu}}_A \\ &= \left(q_d \boldsymbol{U}_d \boldsymbol{Q}_d^{-1} \boldsymbol{U}_d^T + \boldsymbol{I} - \boldsymbol{U}_d \boldsymbol{U}_d^T\right) \hat{\boldsymbol{\mu}}_A \\ &= q_d \boldsymbol{U}_d \boldsymbol{Q}_d^{-1} \boldsymbol{U}_d^T \hat{\boldsymbol{\mu}}_A + \hat{\boldsymbol{\mu}}_A - \boldsymbol{U}_d \boldsymbol{U}_d^T \hat{\boldsymbol{\mu}}_A \end{aligned} \tag{4.7}$$

The same calculation is done for $\boldsymbol{\mu}_B$. Using these quantities we estimate the vector $\boldsymbol{f}$ spanning the Fisher subspace, as follows: $\boldsymbol{f} = \frac{\boldsymbol{\mu}_A - \boldsymbol{\mu}_B}{\|\boldsymbol{\mu}_A - \boldsymbol{\mu}_B\|}$. To classify an unknown point $\boldsymbol{p}$, we transform it with $\boldsymbol{W}$, and we project it on $\boldsymbol{f}$; both these steps are performed by the inner product $\boldsymbol{w} \cdot \boldsymbol{p}$, where:

$$\boldsymbol{w} = \boldsymbol{W}^T \boldsymbol{f} = q_d \boldsymbol{U}_d^T \boldsymbol{Q}_d^{-1} \boldsymbol{U}_d \boldsymbol{f} + \boldsymbol{f} - \boldsymbol{U}_d^T \boldsymbol{U}_d \boldsymbol{f} \tag{4.8}$$

Finally, given γ as in Equation (3.17), $\boldsymbol{p}$ is assigned to class A if $\boldsymbol{w} \cdot \boldsymbol{p} < \gamma$, to class B otherwise.

Notice that we never explicitly compute the matrix $\boldsymbol{W}$, but we perform the matrix times vector operations reported in Equations (4.7,4.8), thus preventing a quadratic time and space complexity. After the training phase, the classification model is represented by $\boldsymbol{w}$ and γ. The described method is a batch classifier that will be referred as truncated `IPCAC` (`TIPCAC`) in the following.

When the cardinality of the training set is too high, or when mini-batches of training data $\boldsymbol{\mathcal{B}}_k = \boldsymbol{\mathcal{B}}_{A,k} \cup \boldsymbol{\mathcal{B}}_{B,k}$ are dynamically supplied, subsequent training phases must be applied to update the classification model. To this aim, `TIPCAC` has been extended to perform **online/incremental** training by updating the following parameters:

$N_k, N_{A,k}, N_{B,k}$: number of training points seen until the k-th training phase;

$\boldsymbol{\mu}_k, \hat{\boldsymbol{\mu}}_{A,k}, \hat{\boldsymbol{\mu}}_{B,k}$: the means employed to obtain the centered sets $\boldsymbol{\mathcal{P}}_k$, $\boldsymbol{\mathcal{P}}_{A,k}$, and $\boldsymbol{\mathcal{P}}_{B,k}$ respectively;

$\boldsymbol{U}_{d_k}, \boldsymbol{Q}_{d_k}, \boldsymbol{V}_{d_k}$: the `SVD` matrices related to $\boldsymbol{\mathcal{P}}_k$, truncated to d_k principal components;

σ_A, σ_B : the standard deviations of the projections $\boldsymbol{w}_k^T \boldsymbol{P}_{A,k}$ and $\boldsymbol{w}_k^T \boldsymbol{P}_{B,k}$.

Consider the case when $k-1$ training phases have been performed and a new mini-batch $\boldsymbol{\mathcal{B}}_k$ is provided, where $|\boldsymbol{\mathcal{B}}_k| = n_k = n_{A,k} + n_{B,k}$. Firstly, we update $N_k, N_{A,k}, N_{B,k}$, $\boldsymbol{\mu}_k, \hat{\boldsymbol{\mu}}_{A,k}, \hat{\boldsymbol{\mu}}_{B,k}$, and we center the points in $\boldsymbol{\mathcal{B}}_k$ around the old mean: $\bar{\boldsymbol{\mathcal{B}}}_k = \{\boldsymbol{p} - \boldsymbol{\mu}_{k-1} | \boldsymbol{p} \in \boldsymbol{\mathcal{B}}_k\}$. Secondly, we update the TSVD matrices [3] by means of the algorithm described in [7] and by exploiting the information carried by $\bar{\boldsymbol{\mathcal{B}}}_k$, so that: $\bar{\boldsymbol{P}}_k \simeq \boldsymbol{U'}_{d_k} \boldsymbol{Q'}_{d_k} \boldsymbol{V'}_{d_k}^T$, where $\bar{\boldsymbol{P}}_k$ represents the set $\bar{\boldsymbol{\mathcal{P}}}_k = \boldsymbol{\mathcal{P}}_{k-1} \cup \bar{\boldsymbol{\mathcal{B}}}_k$. Finally, considering that the updated TSVD matrices are related to the points $\bar{\boldsymbol{\mathcal{P}}}_k$ that are centered on $\boldsymbol{\mu}_{k-1}$, a re-centering operation is required to obtain TSVD matrices related to points centered on $\boldsymbol{\mu}_k$; this is done by applying the re-centering rank-one modification described in [7], choosing as translation the quantity $\frac{n}{N} \langle \bar{\boldsymbol{\mathcal{B}}}_k \rangle$. Given the updated means and TSVD matrices, we can:

- estimate the whitened means $\boldsymbol{\mu}_{A,k}$ and $\boldsymbol{\mu}_{B,k}$ by employing Equation (4.7);
- obtain the updated vector $\boldsymbol{f}_k$;
- compute the the new vector $\boldsymbol{w}_k$ through Equation (4.8).

Notice that these computations require to store only $O(Dn_k + Dd_k)$ real values per mini-batch.

Regarding the update of the thresholding value γ_k we have not employed Equation (3.17), since it requires to store the whole training set, and it is not able to handle unbalanced classes; therefore, we have chosen to set the value of γ_k so that it corresponds to the point having the same Mahalanobis distance $|\xi|$ from the projections of the mean vectors on the Fisher subspace, computed as: $\bar{\mu}_{A,k} = \boldsymbol{f}_k \cdot \boldsymbol{\mu}_{A,k}$ and $\bar{\mu}_{B,k} = \boldsymbol{f}_k \cdot \boldsymbol{\mu}_{B,k}$.
More precisely, we impose that $\bar{\mu}_{A,k} + \xi\, \sigma_{A,k} = \bar{\mu}_{B,k} - \xi\, \sigma_{B,k}$, where $\sigma_{A,k}$ and $\sigma_{B,k}$ are the standard deviations of the projections of the whitened points on $\boldsymbol{f}_k$. Defining $\gamma_k = \bar{\mu}_{A,k} + \xi\, \sigma_{A,k}$ we obtain:

$$\gamma_k = \bar{\mu}_{A,k} + \frac{\sigma_{A,k}\,(\bar{\mu}_{B,k} - \bar{\mu}_{A,k})}{\sigma_{A,k} + \sigma_{B,k}}$$

[3] We underline that the number of retained components $d_k = \min(\log_2^2 N_k, D)$ might increase at each step, so that the rank of the computed TSVD matrices would grow accordingly, up the maximum value D.

Considering that $\sigma^2 = \mathbb{E}\left[x^2\right] - \mathbb{E}\left[x\right]^2$, the employed quantities are updated as follows:

$$\begin{aligned}
\bar{\mu}_{A,k} &= \frac{N_{A,k-1}\ \bar{\mu}_{A,k-1} + n_{A,k}\left\langle \boldsymbol{w}_k^T \boldsymbol{\mathcal{B}}_{A,k}\right\rangle}{N_{A,k}} \\
\bar{\mu}_{2A,k-1} &= \mathbb{E}\left[\left\{\left(\boldsymbol{w}_k^T \boldsymbol{p}\right)^2 \mid \boldsymbol{p} \in \boldsymbol{\mathcal{P}}_{A,k-1}\right\}\right] = \bar{\mu}_{A,k-1}^2 + \sigma_{A,k-1}^2 \\
\bar{\mu}_{2A,k} &= \mathbb{E}\left[\left\{\left(\boldsymbol{w}_k^T \boldsymbol{p}\right)^2 \mid \boldsymbol{p} \in \boldsymbol{\mathcal{P}}_{A,k}\right\}\right] \\
&= \frac{N_{A,k-1}\ \bar{\mu}_{2A,k-1} + n_{A,k}\left\langle \left(\boldsymbol{w}_k^T \boldsymbol{\mathcal{B}}_{A,k}\right)^2\right\rangle}{N_{A,k}} \\
\sigma_{A,k} &= \sqrt{\bar{\mu}_{2A,k} - \bar{\mu}_{A,k}^2}
\end{aligned}$$

where we have defined the set of real values

$$\left(\boldsymbol{w}_k^T \boldsymbol{\mathcal{B}}_{A,k}\right)^2 = \{(\boldsymbol{w}_k \cdot \boldsymbol{p})^2 | \boldsymbol{p} \in \boldsymbol{\mathcal{B}}_{A,k}\}$$

The updated quantities $\bar{\mu}_{B,k}$, $\bar{\mu}_{2B,k}$, and $\sigma_{B,k}$ are similarly evaluated.

After the k-th update of the classification model, the described online algorithm approximates the Fisher subspace with respect to the points in $\boldsymbol{\mathcal{P}}_k$, and these points will no longer be needed for future updates; therefore, O-IPCAC performs only one pass through the data to compute its classification model.

The computational cost of the training phase is dominated by the incremental rank-d SVD that requires $O(DNd)$ operations for $d \leq \sqrt{\min(N, D)}$ [7]. Regarding the memory requirements, O-IPCAC stores $O(Dn_k + Dd_k)$ real values during the training tasks, and only $O(D)$ values during classification.

4.3.1 An Adaptive Version of OIPCAC

In this subsection we describe how to handle a probability distribution underlying the data that changes with time; to this aim, we maintain the information related to a fixed budget of the most recently supplied training vectors in an on-line fashion, and suppress the information related to the oldest training vectors.

As reported in [7], and explained in Section 4.2, given the following TSVD decomposition:

$$\boldsymbol{\mathcal{P}}_k = [\boldsymbol{\mathcal{B}}_1 \cdots \boldsymbol{\mathcal{B}}_k] \simeq \boldsymbol{U}_d \boldsymbol{Q}_d \boldsymbol{V}_d^T$$

it is possible to update it to get a new TSVD decomposition:

$$\bar{\boldsymbol{U}}_d \bar{\boldsymbol{S}}_d \bar{\boldsymbol{V}}_d^T \simeq [\boldsymbol{\mathcal{B}}_2 \cdots \boldsymbol{\mathcal{B}}_{k+1}] = \boldsymbol{\mathcal{P}}_{k+1} \tag{4.9}$$

such that the contributions given by the oldest points in $\boldsymbol{\mathcal{B}}_1$ is inhibited, and the information related to novel training vectors in $\boldsymbol{\mathcal{B}}_{k+1}$ is added.

To obtain an adaptive version of O-IPCAC we update the vector $\boldsymbol{w}$ by employing Equation (4.8), and by using the updated TSVD matrices to perform the update, downdate, and recentering modifications proposed in Section 4.2.
Subsequently, we need to modify the threshold γ through the updated means $\boldsymbol{\mu}_{k+1}, \boldsymbol{\mu}_{A,k+1}, \boldsymbol{\mu}_{B,k+1}, \bar{\mu}_{A,k+1}, \bar{\mu}_{2A,k+1}, \bar{\mu}_{2B,k+1}$, the updated standard deviations $\sigma_{A,k+1}, \sigma_{B,k+1}$, and the $N_{k+1}, N_{A,k+1}, N_{B,k+1}$ values. We modify these quantities both by considering the novel training vectors and by inhibiting the information given by those in the oldest training mini-batch.

The update of $N_{k+1}, N_{A,k+1}, N_{B,k+1}$ is straightforward, while the other quantities are computed as follows:

$$\begin{aligned}
\boldsymbol{\mu}_{k+1} &= \frac{N_k \boldsymbol{\mu} + N_{\boldsymbol{\mathcal{B}}_{k+1}} \boldsymbol{\mu}_{\boldsymbol{\mathcal{B}}_{k+1}} - N_{\boldsymbol{\mathcal{B}}_1} \boldsymbol{\mu}_{\boldsymbol{\mathcal{B}}_1}}{N + N_{\boldsymbol{\mathcal{B}}_{k+1}} - N_{\boldsymbol{\mathcal{B}}_1}} \\
\boldsymbol{\mu}_{A,k+1} &= \frac{\boldsymbol{\mu}_{A,k} N_{A,k} + N_{A,\boldsymbol{\mathcal{B}}_{k+1}} \boldsymbol{\mu}_{A,\boldsymbol{\mathcal{B}}_{k+1}} - N_{A,\boldsymbol{\mathcal{B}}_1} \boldsymbol{\mu}_{A,\boldsymbol{\mathcal{B}}_1}}{N_A + N_{A,\boldsymbol{\mathcal{B}}_{k+1}} - N_{A,\boldsymbol{\mathcal{B}}_1}} \\
\bar{\mu}_{A,k+1} &= \frac{N_{A,k} \bar{\mu}_{A,k} + N_{A,\boldsymbol{\mathcal{B}}_{k+1}} \bar{\mu}_{A,\boldsymbol{\mathcal{B}}_{k+1}} - n_{A,\boldsymbol{\mathcal{B}}_1} \bar{\mu}_{A,\boldsymbol{\mathcal{B}}_1}}{N_{A,k} + N_{A,\boldsymbol{\mathcal{B}}_{k+1}} - n_{A,\boldsymbol{\mathcal{B}}_1}} \\
\bar{\mu}_{2A,k+1} &= \frac{N_{A,k} \bar{\mu}_{2A,k+1} + N_{A,\boldsymbol{\mathcal{B}}_{k+1}} \bar{\mu}_{2A,\boldsymbol{\mathcal{B}}_{k+1}} - N_{A,\boldsymbol{\mathcal{B}}_1} \bar{\mu}_{2A,\boldsymbol{\mathcal{B}}_1}}{N_{A,k} + N_{A,\boldsymbol{\mathcal{B}}_{k+1}} - N_{A,\boldsymbol{\mathcal{B}}_1}} \\
\sigma_{A,k+1} &= \sqrt{\bar{\mu}_{2A,k+1} - \bar{\mu}_{A,k+1}^2}
\end{aligned}$$

where $\boldsymbol{\mathcal{B}}_{k+1}$ is the subscript referred to the new mini-batch of points, and $\boldsymbol{\mathcal{B}}_1$ is the oldest training mini-batch that must be inhibited; moreover, N_* are the cardinalities, μ_* are the means, $\bar{\mu}_{A,*}$ and $\sigma_{A,*}$ are the projected means and standard deviation on $\boldsymbol{w}$. The updated quantities for the class B are similarly evaluated.

4.3.2 Experiment to evaluate the d-Dimension

In this section we describe the experiment on synthetic datasets that we performed to evaluate the relation between the classification performance of

O-IPCAC and the number d of considered eigenvalues. We recall that O-IPCAC considers only the largest eigenvalues, substituting the whitening step by a partial whitening process that whitens the training vectors projected along the first $d \leq D$ principal components, and keeps the remaining components unchanged.

The points in the synthetic dataset have been randomly drawn from two non-isotropic MGDs.

For each MGD, the mean vector coordinates were randomly selected in the open interval $(0, 1)$ according to the uniform distribution, and then multiplied by 0.01 **to reduce** the separation between the classes, while the covariance matrices were obtained by randomly generating both the positive eigenvalues, and the orthonormal eigenvectors. More precisely, the eigenvalues were generated by employing the random generator for the uniform distribution, while the eigenvectors were drawn from the normal distribution, and they were orthogonalized through the Gram-Schmidt orthogonalization process.

By employing this generator we produced 100 datasets ($\boldsymbol{\mathcal{D}}_i = \{\boldsymbol{x}_j\}_{j=1}^{N}$, $i \in \{1, \cdots, 100\}$, $\boldsymbol{x}_j \in \Re^{500}$), each composed by 500-dimensional vectors split into two classes with 250 points per class; we note that this datasets are characterized by a ratio between the number of points and the space dimensionality equal to one.

Under this experimental setting we could ascertain that the behavior of the accuracy as function of d is described by the following equation:

$$F(d) = \frac{\sum_{i=1}^{100} \left(\frac{1}{10} \sum_{k=1}^{10} Accuracy\left(\boldsymbol{\mathcal{D}}_i^k(d)\right) \right)}{100}$$

where $Accuracy\left(\boldsymbol{\mathcal{D}}_i^k(d)\right)$ is the accuracy achieved on the $\boldsymbol{\mathcal{D}}_i$ dataset in the k^{th} fold. In Figure 4.2 the results achieved are summarized. Note that the performance of O-IPCAC is stable for each value of d that obliges the whitening process to neglect the 50 smallest principal components. The best results are achieved on the interval $[70, 100]$ that contains the value $d = \min(\log_2^2 N, D)$; this experiment confirms the choice of d reported in Section 4.3. We further note that a similar behavior was observed in large-scale latent semantic indexing, as reported in [6].

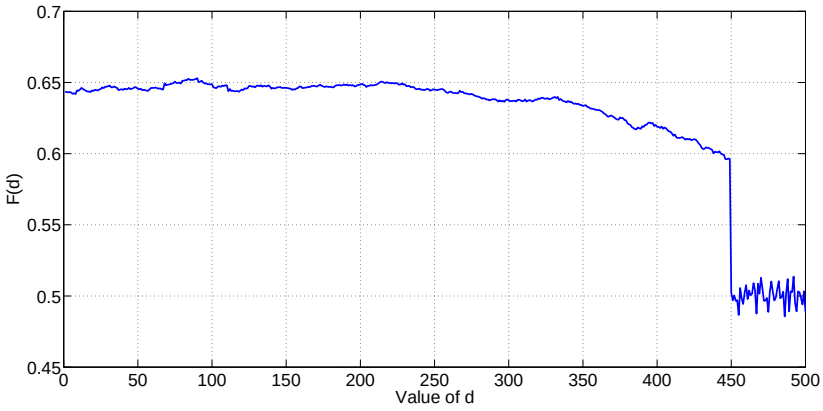


Figure 4.2: Averaged results on 100 datasets employing 10−fold cross validation in function of the value of d.

4.4 Results

In the following we firstly describe the experimental settings (see Section 4.4.1), and the tests performed on both synthetic data (see Section 4.4.2) and real data (see Section 4.4.3 and Section 4.5). To test the efficacy of the proposed techniques, we compare the obtained results to those computed by well known classification methods. More precisely, we consider as base-line algorithms the following batch algorithms: `IPCAC`, `IPCAC`$_{var}$ (`IPCAC` maintaining a fixed amount of variance[4]), `FLDA`, and `SVM`. Furthermore, we compare our method with base-line online methods that are: Perceptron [80], Projectron and its improvement [72], Passive-Aggressive (`PA-I`, [27]), Online Independent Support Vector Machines (`OISVM`, [71]), and the Second Order Perceptron (`SOP`, [11]) implemented in [70]. Finally, in Section 4.4.6 we also report the test that allowed us to compare the performance achieved by `OIPCAC` and `R-FLDA`, when the test dataset has a cardinality which is lower than the space dimensionality, hereby causing the small sample size problem to occur.

For all the tested algorithms, we selected the linear kernel where needed, and we tuned the other parameters by using the grid optimization with 10-fold cross-validation on a randomly selected subset of the training data. All

[4]We maintained the 99% of the variance.

the tested algorithms are implemented in MATLAB[5].

4.4.1 Experimental Setting

To test the performances of our classifiers, we executed two kinds of evaluation:

K-folding: we adopted this method to evaluate the performances of the compared algorithms. When testing online algorithms, we executed the training phase by performing iterative updates with the data contained on $K-1$ folds. To obtain a statistically significant evaluation, for each classifier we estimated the classification accuracy for all the K folds, and we executed R runs of K-folding permuting the data, thus obtaining KR accuracies to be averaged.

Average Online Error rate: to assess the efficacy of online training algorithms, we also evaluated the average online error. More precisely, for each point $\boldsymbol{p}_t$ given at time t, $\boldsymbol{p}_t$ was at first used to test the classifier; after classification, the evaluated average online error at time t was computed as the cumulative mistakes divided by the total observed instances:

$$Err(t) = \frac{\sum_{k=1}^{t} err(k)}{t}$$

where $err(k) = 1$ if the classifier has wrongly classified $\boldsymbol{p}_k$, $err(k) = 0$ otherwise. After the evaluation of $Err(t)$, $\boldsymbol{p}_t$ was employed to update the classification model. This method is also called the learning curve and let us compare the learning speed of O-IPCAC with that of the other online classifiers.

4.4.2 Synthetic Datasets

The synthetic datasets, were generated employing the generator described in Section 4.3.2.

By employing this generator we produced 2 datasets, each composed of two classes with 1200 points per class in $\Re^{2000}$, thus maintaining the ratio

[5]For the SVM classifier we used a MATLAB wrapper to the library "libSVM" [12].

between the number of points and dimensions approximately equal to one. Notice that the second dataset differs from the first one since we reduced the distance between the classes by multiplying the class means by 0.05.

4.4.3 Real Datasets

The three real datasets employed for testing the proposed techniques are:

SpamAssassin: it is composed by 900 spam e-mails and 900 legitimate e-mails, extracted from the SpamAssassin corpus [1]. As described in Section 3.7, for each email we calculated an high dimensional, and sparse feature vector containing the word occurrences with respect to a dictionary composed by 87000 entries, and we applied the Term Frequency-Inverse Document Frequency (`TF-IDF`) weighting schema. Moreover, we employed the Latent Semantic Analysis (`LSA`) technique to compute a sparse to dense projection matrix that maps the feature vectors on a dense subspace, thus obtaining a set of 1800 points in $\Re^{1004}$.

Hungarian heart diseases: it is composed by 294 feature vectors in $\Re^{13}$ extracted from the UCI machine learning repository [2]. More precisely, 106 points are related to patients affected by heart disease, and 188 points are related to healthy patients. This is an interesting dataset for at least two reasons. First of all, the probability distribution underlying each class is strongly different from a `MGD`, so that the results achieved on this dataset reflect the dependency of `IPCAC` based classifiers from their base assumption. Secondly, this dataset is very small compared with all the other datasets; this fact stresses the learning speed of online algorithms.

Reuters Corpus Volume I: it is composed by 800000 manually categorized newswire stories made available by Reuters for research purposes. To obtain a two-class classification problem we defined the class A to be the one containing all the documents tagged with the label *Corporate*, whilst the class B contains all the other documents. From the obtained dataset we randomly selected 23149 training vectors and

199328 test vectors in $\Re^{47236}$, where 108809 points belong to class A and 113668 belong to class B.

4.4.4 Experimental Comparison with Batch Algorithms

We evaluated O-IPCAC by testing it on the synthetic datasets, and on the SpamAssassin and Hungarian heart diseases datasets, performing 10 runs of 10-fold cross validation. Moreover, we compared its results with those achieved by well-known batch methods.

Classifier	Synth 1	Synth 2	SpamAssassin	Hungarian
O-IPCAC	**99.80**% (±0.0030)	**90.61**% (±0.0174)	94.03% (±0.0152)	80.53% (±0.0709)
IPCAC$_{var}$	97.50% (±0.0068)	83.28% (±0.0269)	**94.61**% (±0.0152)	66.44% (±0.0664)
IPCAC	82.23% (±0.0266)	58.01% (±0.0353)	86.44% (±0.0246)	**81.33**% (±0.0638)
FLDA	81.28% (±0.0279)	55.89% (±0.0332)	86.10% (±0.0218)	81.09% (±0.0623)
SVM	99.10% (±0.0037)	87.91% (±0.0220)	86.76% (±0.0244)	80.35% (±0.0692)

Table 4.2: Experimental results (accuracy%, (±std)) obtained by comparing O-IPCAC with batch methods.

Observing the results shown in Table 4.2, we note that O-IPCAC outperforms the other methods on the synthetic datasets. In these tests, characterized by a ratio between the number of points and the space dimensionality similar or smaller than one, SVM has a slightly worse performance, IPCAC and FLDA are affected by serious decrease of the accuracy, whilst IPCAC$_{var}$ suffers for the information loss caused by the dimensionality reduction step.

Considering the real datasets, IPCAC and FLDA achieve the best results on the Hungarian dataset, where there is a sufficient amount of points with respect to the space dimensionality. Note that, the results achieved on this dataset are useful to demonstrate the importance of the residuals to maintain all the discriminative information. Indeed, although more than the 99% of variance is contained in the first 4 principal components, IPCAC$_{var}$ obtains an accuracy of 66.44%. On the other side, O-IPCAC tested by applying 10-fold cross validation with $d = 4$ obtained an higher accuracy (79.33% with respect to the 66.44 obtained by IPCAC$_{var}$). Finally, we can underline that, when O-IPCAC is not the best performing classifier its results are anyway comparable with the best ones.

The overall results confirm what follows:

1. performing the "full" whitening process, by employing also the eigenvectors related to the smallest eigenvalues of the covariance matrix, might produce decrease of the classification performance when the ratio between the space dimension and the cardinality of the training set is approximately equal to one;

2. dimensionality reduction might produce information loss that affects the classification results.

As shown by our tests, `O-IPCAC` demonstrates to be a good solution when the ratio between the space dimensionality and the training set cardinality is ≈ 1, since it does not apply any dimensionality reduction technique that might discard discriminative information.

4.4.5 Experimental Comparison with Online Algorithms

To further evaluate the efficacy of `O-IPCAC` we compared it with well-known online classifiers. Note that in these tests we employed also the Reuters dataset. We note that this dataset has not been used to test the batch algorithms since its high cardinality and high space dimensionality make the application of several batch classifiers impractical.

To test the algorithms we applied 10 runs of 10-fold cross validation as described in Section 4.4.1 and we plotted the curves of the average online error.

Classifier	Synth 1	Synth 2	SpamAssassin
`O-IPCAC`	**99.80**% (±0.0030)	**90.61**% (±0.0174)	94.03% (±0.0152)
`PA-I`	99.71 (±0.0037)	88.04% (±0.0200)	75.96% (±0.1168)
Perceptron	97.05 (±0.0106)	82.43% (±0.0256)	65.57% (±0.1149)
Projectron	96.88% (±0.0119)	82.64% (±0.0265)	64.03% (±0.1252)
Projectron++	96.99% (±0.0124)	82.56% (±0.0278)	74.22% (±0.1183)
`OISVM`	99.70% (±0.0036)	88.32% (±0.0199)	88.33% (±0.0257)
`SOP`	97.17% (±0.0122)	82.13% (±0.0276)	91.48% (±0.0264)

Table 4.3: Experimental results (accuracy%, (±std)) obtained by employing online algorithms.

Classifier	Hungarian	Reuters
O-IPCAC	80.53% (±0.0709)	**93.77**% (±0.0038)
PA-I	53.06% (±0.1622)	91.19% (±0.0134)
Perceptron	54.41% (±0.1433)	87.66% (±0.0363)
Projectron	63.00% (±0.1045)	92.25% (±0.0061)
Projectron++	60.37% (±0.1423)	88.85% (±0.0266)
OISVM	**81.23**% (±0.0673)	—
SOP	78.47% (±0.0744)	91.22% (±0.0065)

Table 4.4: Experimental results (accuracy%, (±std)) obtained by employing online algorithms.

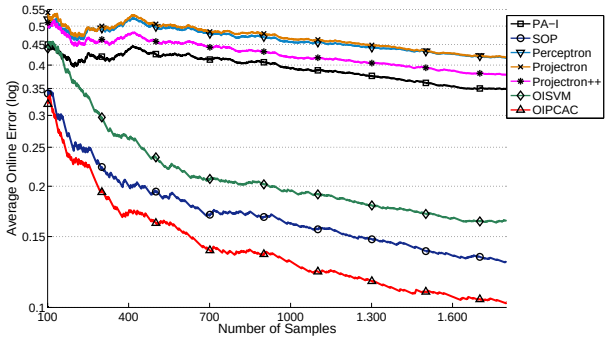


Figure 4.3: Average training error curves on SpamAssassin Corpus. Y axis is in log scale.

As reported in Table 4.3 and Table 4.4, O-IPCAC obtains the best results outperforming the other online techniques when tested on synthetic, SpamAssassin, and Reuters datasets. Moreover, in our tests, OISVM failed to manage the high cardinality and dimensionality of the Reuters dataset.

Considering the average online errors plotted in Figure 4.3 and Figure 4.4, it is possible to notice the different learning behavior of second order methods and first order techniques. In fact, the second order algorithms need a smaller number of training vectors to achieve good classification performances. Nevertheless, with a huge amount of data, PA-I obtains good results, as shown in Figure 4.5. Finally, it is important to underline that O-IPCAC achieved the best learning speed on the Reuters and SpamAssasin datasets, whilst OISVM outperforms our method on the Hungarian dataset due to its high "non-Gaussianity".

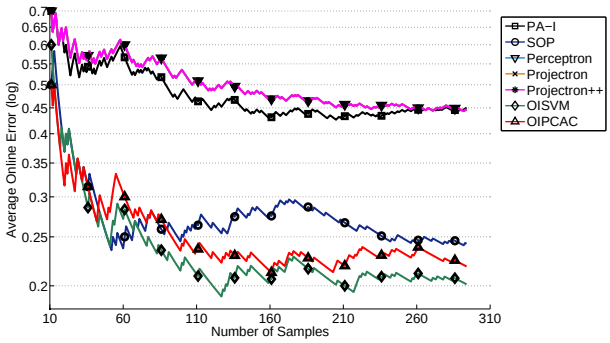


Figure 4.4: Average training error curves on Hungarian dataset. Y axis is in log scale. Note that the curves related to Perceptron, Projectron, and Projectron++ are overlapped.

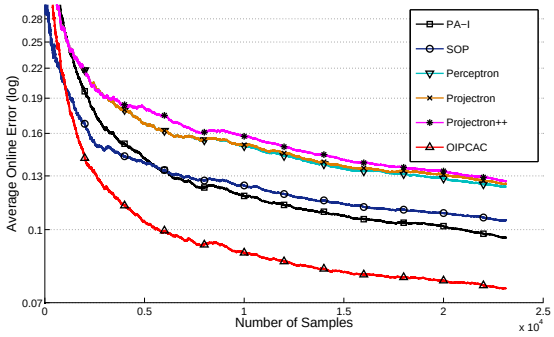


Figure 4.5: Average training error curves on Reuters dataset. Y axis is in log scale.

4.4.6 Comparison beetween R-FLDA and OIPCAC

R-FLDA was developed to deal with the small simple size problem; more precisely, this method is more robust with respect to the other FLDA based techniques, when the cardinality of the dataset is lower than its space dimensionality. For this reason, in this section we have compared O-IPCAC with R-FLDA under this condition. To this aim, we have employed the same generator described in Section 4.3.2 to generate two synthetic datasets with two classes, where the points in each class are drawn from a randomly selected non-isotropic MGD. The first dataset is composed of two classes with 600 points per class in $\Re^{2000}$, thus maintaining the ratio between number of points and dimensionality approximately equal to one. The second dataset

was produced reducing the distance between the classes of the first dataset, multiplying the class means by 0.1

We have applied R-FLDA selecting the threshold by employing the Equation (3.17), and we have performed a grid optimization to empirically select the best values of the required parameters.

Classifier	Synth 1	Synth 2
O-IPCAC	**77.67**% (±0.0342)	**74.92**% (±0.0462)
R-FLDA	60.33% (±0.0560)	57.87% (±0.0542)
IPCAC	50.21% (±0.0678)	49.25% (±0.0556)
IPCAC$_{var}$	71.75% (±0.0394)	70.92% (±0.0495)

Table 4.5: Experimental results (accuracy%, (±std)) obtained by comparing O-IPCAC with R-FLDA employing synthetic datasets with cardinality lower than dimensionality.

In Table 4.5 the obtained results are summarized, showing that O-IPCAC obtains the best results, outperforming the other techniques; moreover, considering the accuracies achieved by R-FLDA it is possible to notice that our method is more robust to the small sample size problem, and it does not need a computationally expensive tuning phase.

4.5 A Real Application: EEG Classification

Recently EEG classification is raising a wide interest since it is the fundamental step of Brain to Computer Interface (BCI) systems, which are based on the translation of brain activity into commands for computers. Although a great deal of research work has been devoted to the task of EEG classification, and several methods have been proposed in the literature (and reviewed in [61]), the problem is still open. Indeed, the task of EEG classification is an hard problem, since the two classes are often highly unbalanced, the selection of discriminative information is difficult, the data are high dimensional, and the cardinality of the training sets is often lower than their dimensionality. To deal with these problems, feature extraction/selection techniques are generally used to compute a small number of features representing the data; unfortunately, this approach causes loss of discriminative information, and might affect the classification accuracy. Note that, while dimensionality

reduction is exploited as the first pre-processing step of several EEG classification systems [98], so that their performance mainly depends on the quality of the used features, O-IPCAC can be applied on the raw data since it has been developed to deal with high dimensional datasets whose cardinality is lower than the space dimensionality.

The EEG dataset employed in this work has been distributed for the MLSP 2010 competition [48]; it consists in a multi-channel time-series containing measures of brain electrical activity recorded while a subject viewed satellite images. The classifier must analyze the brain activity to recognize those images containing a predefined target.

The promising results achieved in the MLSP competition, and the comparison with state-of-the-art methods, demonstrate the efficacy of our approach.

This section is organized as follows: in Section 4.5.1 the training EEG data and the classification problem are described, in Section 4.5.2 the experimental setup is summarized, and the achieved results are reported.

4.5.1 Data Description and Pre-processing

The data used in our tests have been distributed by the organizers of the MLSP 2010 competition for research purposes[6], and consist of EEG brain signals collected while the subject viewed satellite images and tried to detect those containing a predefined target. There are 64 channels of EEG data, the total number of samples is 176378, and the sampling rate is 256Hz. During the EEG recording 2775 satellite images were shown, partitioned in 75 activation blocks with 37 images per block. All images within a block were consecutively displayed for 100 ms (an "image trigger" is provided to indicate the time samples corresponding to the on-set of each image). Each block was initiated by the subject after a rest period, the length of which was not specified in advance. The classifier must analyze the brain activity to recognize those images containing the target.

Before applying OIPCAC we pre-processed each channel with a Gaussian filter with cut-frequency of 2.2Hz, and we subtracted the filtered data from the original one to obtain high-pass filtered signals. These signals were

[6] http://www.bme.ogi.edu/~hildk/mlsp2010Competition.html.

then used to extract 64×97 image blocks, where each image block starts exactly 65 time samples ($\approx$ 250ms) after the corresponding image trigger. We underline that each image block covers a time window approximately located between 250ms and 550ms after the image trigger, since this range contains the P300 waves [76] and other possible brain activations [14]. The extracted blocks are serialized in 2775 vectors in $\Re^{6208}$, since only 58 points represent images with target, this dataset is highly unbalanced.

4.5.2 Experimental Results

In this section we consider both the results achieved in the MLSP 2010 competition [48], and the tests we performed to compare O-IPCAC with state-of-the-art online/incremental classifiers. It is important to underline that the high dimensional EEG data considered in this work cannot be processed (by 32 bits PCs) by most batch algorithms such as FLDA or SVM, due to either too high memory requirements, or too long training time. On the other side, online algorithms such as O-IPCAC can handle this problem since they perform subsequent training phases on mini-batches of training data. Therefore we compare our method with: Perceptron, Second Order Perceptron (SOP, [11]), Online Independent SVM (OISVM, [71]), Passive Aggressive (PA, [27]), Alma [41], and Incremental FLDA (ILDA, [56])[7]. All the tested algorithms are implemented in MATLAB[8];

To evaluate the performance of our classifier comparing it with the other methods, we employed the dataset described in Section 4.5.1, we computed the Receiver Operating Characteristic (ROC) curve, and we estimated the Area Under the Curve (AUC, see Appendix A). To obtain an unbiased evaluation, we performed ten-fold cross validation, and we averaged the computed sensitivity and specificity values.

Figure 4.5.2 and Table 4.5.2 show the obtained ROC curves and the AUC of

[7] To tune all the parameters of the employed algorithms we employed 2-fold cross validation, and we selected the configuration that achieved the best accuracy. Notice that for the kernel methods (SOP, OISVM, PA, Alma) the best results are achieved by choosing the linear kernel.

[8] For the SVM classifier we used a MATLAB wrapper to the library "libSVM" [12]; for all the online algorithms, except ILDA, we used a MATLAB implementation called dogma [70], while for ILDA based classifier we have used the authors' implementation avaible at http://mi.eng.cam.ac.uk/∼tkk22/code.htm.

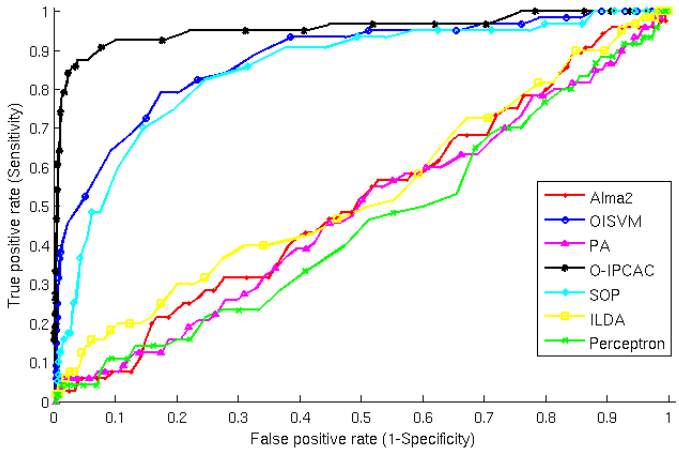


Figure 4.6: ROC curves

the tested classifiers; it can be noted that O-IPCAC achieves the best results. Furthermore, these results demonstrate that the first order techniques (Perceptron, Alma, and PA) cannot discriminate the two classes, while O-IPCAC, SOP, and OISVM, which are second order techniques, achieve good results. Note that ILDA, despite being a second order method, obtains bad results since it is not able to manage datasets whose cardinality is lower than their dimensionality. We underline that our method is able to handle strongly unbalanced classes thanks to the thresholding method based on the Mahalanobis distance that avoids any experimental setup.

Regarding the MLSP competition, the organizers have evaluated the various approaches by employing a test set that differs from the previously described training set (see Section 4.5.1). Indeed, during the experiments performed to acquire the test data, different image durations (50ms, 100ms, 150ms, and 200ms) are used in different activation blocks. While the training set was distributed by the MLSP organizers, the test set was not.

In the MLSP competition the efficacy of the proposed approaches were evaluated by estimating the ROC curves, and comparing the AUCs. Our algorithm covered the 80% of the AUC, ranking seventh among the 35 participants [48]. It is worth noting that the first 10 algorithms have very close classification performance (between 82% and 79%). Among the methods participating to the competition, those based on FLDA achieved the best av-

Classifier	AUC
O-IPCAC	**0.9541**
OISVM	0.8766
SOP	0.8479
ILDA	0.5315
Alma	0.5110
PA	0.4835
Perceptron	0.4507

Table 4.6: AUC per classifier

erage result (70%), overcoming methods based on SVM, Neural Networks, and Linear Logistic. Considering that we performed just an high-pass filtering as pre-processing step, avoiding any kind of bootstrap aggregating technique, we believe that the achieved results are very promising, and they confirm the good quality of the proposed classifier and its capability to manage classification problem where the dataset cardinality is smaller than the space dimensionality, solving the Small Sample Size problem mentioned in Section 3.2.

4.6 Conclusion

In this chapter we described the Online-IPCAC linear binary classifier (O-IPCAC), an efficient and effective method that performs a "partial" whitening step recovering the residuals. More precisely, the data whitening has been replaced by a process that whitens the data in a linear subspace $\pi_d = \mathbf{Span}\left\langle \boldsymbol{v}_1, \ldots, \boldsymbol{v}_d \right\rangle, d \ll D$, while maintaining unaltered the information related to the orthogonal subspace $(\pi_d)^{\perp} = \mathbf{Span}\left\langle \boldsymbol{v}_{d+1}, \ldots, \boldsymbol{v}_D \right\rangle$.

O-IPCAC has been developed to deal with: high dimensional data, classification problems where the cardinality of the point set is high or the data are dynamically supplied, and highly unbalanced training sets whose cardinality is lower than the space dimensionality.

This technique reduces the effect of small sample size problem and, unlike previous approaches, it considers the case where the number, N, of training examples is approximately equal to the space dimensionality ($N \approx D$). In such circumstances the sample covariance matrix is nonsingular but it still

is not a consistent estimator of the population covariance matrix, as proved in [55].

The classification results achieved by `O-IPCAC` are promising and they outperform those obtained by well-known methods.

Chapter 5

Kernel-IPCAC and Perceptron-IPCAC

The classification algorithms proposed in the previous chapters are promising, but they cannot deal with non-linearly separable classes; to overcome this weakness, in this chapter we present two improvements of IPCAC.

At first we present a method called Kernel-IPCAC (K-IPCAC) and its variants (K-IPCAC$_{var}$ and K-TIPCAC), which are kernel versions of IPCAC exploiting the kernel trick.

Although effective, these classifiers are based on assumptions about the probability distribution function (pdf) underlying the data, so that their performance decrease when the considered dataset does not fulfill the required assumptions. More precisely, IPCAC and their kernel versions cannot cope with the more general case of classes whose underlying pdf is multimodal. To solve this problem, classifiers assuming mixtures of components as class conditional pdf have been proposed two decades ago, and they have been studied by few researchers [73, 37, 42]. To our knowledge, the most important works in this field, which are also related to our approach, are those by Grim *et al.*; they are exhaustively summarized in [42]. In their work, the authors design probabilistic neural networks (PNNs [4]) by assuming that the class conditional distributions are finite mixtures of product components. In particular, they employ a training dataset for each class, estimating the mixture parameters by means of Expectation Maximization (EM, [4]). Furthermore, considering that EM computes mixture parameters closely related to the training data, and that these parameters uniquely de-

termine the performance of the PNNs, in [42] the authors define a way to adapt the network weights to a specific input, and a method to adapt the input to a more probable form. Although the computational cost of the proposed method is too high to be applied in several real problems, the formal proofs reported in [42] are interesting.

For this reason, in Section 5.4 we define a technique, called Perceptron-IPCAC (P-IPCAC), that exploits a similar, but more simpler, idea; indeed, this method employs a neural network to combine different trained linear (IPCAC) classifiers to address classification problems where each class conditional distribution is approximated by one Mixture of Gaussians (MoG). Although this assumption is less general than that of Grim *et al.*, experimental results have shown that the developed method deals efficiently also with real data whose underlying distribution is not a MoG.

This chapter is organized as follows: in Section 5.1 related works are reported; in Section 5.2 K-IPCAC and its improvements are described in details; in Section 5.3 the performances achieved by K-IPCAC on the Spam Assassin datasets are reported; in Section 5.4 P-IPCAC is explained in details; in Section 5.5 experimental results confirming the efficacy of this method are shown.

5.1 Related Works

In this section we summarize the Kernel Principal Component Analysis (see Section 5.1.1) and the Kernel Fisher Discriminant (see Section 5.1.2), which are the kernel versions of PCA and FLDA, respectively. For this reason, they are strongly related to the kernel version of IPCAC.

5.1.1 Kernel Principal Component Analysis

PCA uses only second order statistics in the form of covariance matrix, so that the best it can do is to fit an ellipsoid around the data. Kernel Principal Component Analysis (KPCA, [85]) is an attractive method for extracting non-linear features from a given set of multivariate data. While Principal Component Analysis finds the best ellipsoidal fit for the data, KPCA has the

capability of extracting the non-linear features which could represent the data more naturally (see Figure 5.1).

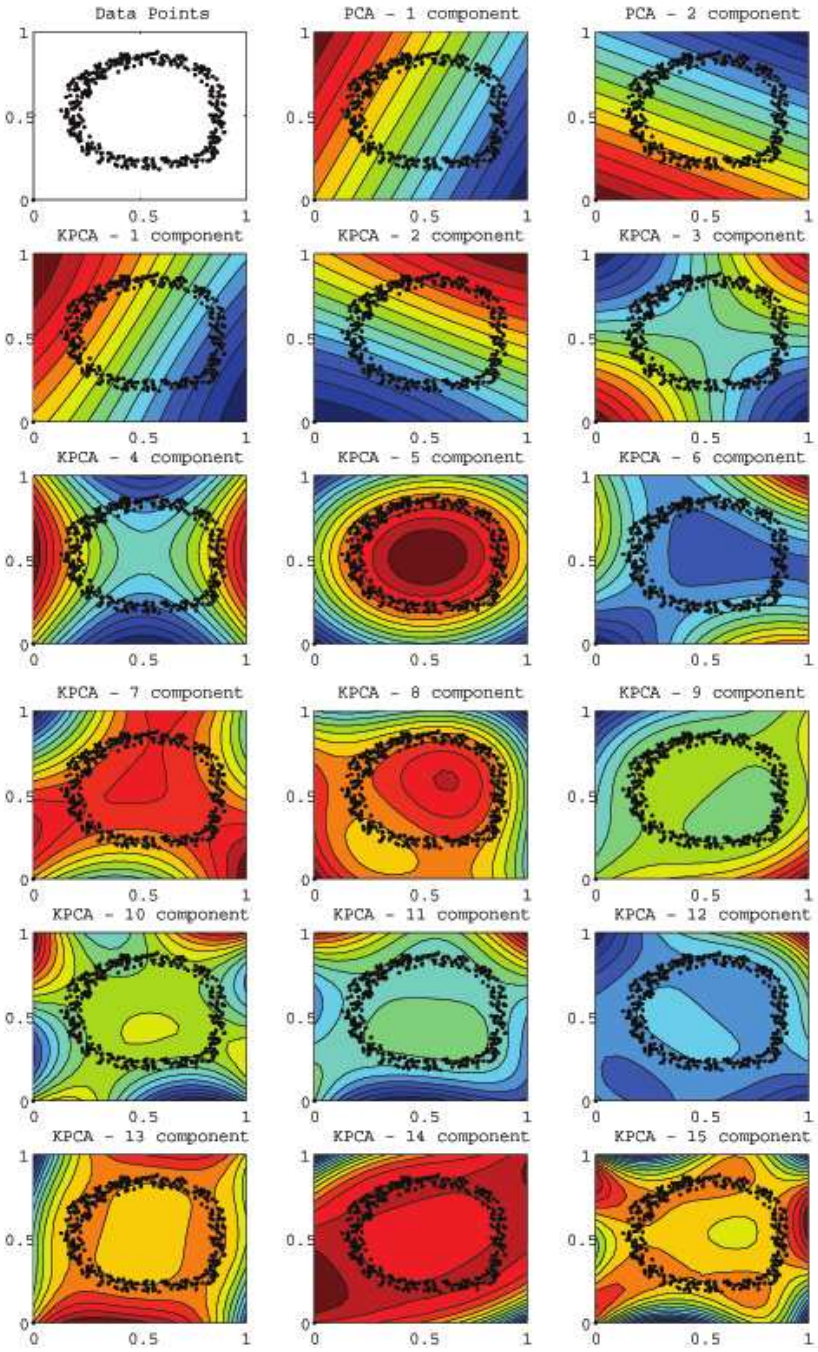


Figure 5.1: Two dimensional example illustrating `KPCA`. This example shows the kernel principal components compared to the principal components.

The essential idea of KPCA is to employ a non linear mapping to project data points into a higher dimensional (possibly infinite) space, where they are represented with non linear features that better describe their peculiarities.

To this aim, the kernel trick [94] is employed since it mitigates the growth of complexity due to the high dimensionality mapping. The kernel trick is based on the Mercer's theorem, which states that any continuous, symmetric, positive semi-definite kernel function $k(\boldsymbol{x}, \boldsymbol{y})$ can be expressed as a dot product in an high dimensional space. More specifically, if the arguments to the kernel are in a measurable space $\boldsymbol{\mathcal{Q}}^D$, and if the kernel is positive semi-definite, i.e. $\sum_{i,j} k(\boldsymbol{x}_i, \boldsymbol{x}_j) c_i c_j \geq 0$ for any finite subset $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ of $\boldsymbol{\mathcal{Q}}^D$ and any real numbers $\{c_1, \cdots, c_n\}$, then a map $\phi : \boldsymbol{\mathcal{Q}}^D \rightarrow \boldsymbol{\mathcal{Q}}_\phi^h$ must exist, whose range is in an inner product space of possibly high dimension, such that

$$k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}).$$

The kernel trick transforms any algorithm that solely depends on the dot product between two vectors in an euclidean space $\Re^D$ in an equivalent algorithm operating in the inner product space $\boldsymbol{\mathcal{Q}}_\phi^h$ implicitly defined by the kernel function $k(\cdot, \cdot)$, that is the range of the map ϕ. Notice that once we have this mapping KPCA is nothing but Linear PCA done on the points in the higher dimensional space. To this aim, wherever a dot product is used in the linear algorithm of interest, it can be replaced with the kernel function, to obtain a linear algorithm operating into a non-linear space. Note that, the range space of ϕ may be very high or even infinite dimensional, this is the reason why the ϕ map is never explicitly computed.

Given N data points in Re^D let $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N$ be D-dimensional column vectors representing them, the algorithmic steps of KPCA could be summarized as follows:

1. Subtract the mean from all the data points.
2. Choose an appropriate kernel $k(\cdot, \cdot)$.
3. Calculate the $N \times N$ Gram matrix $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{ij}$, by applying the kernel operator to the data points.

4. Compute the centered Gram matrix:

$$\tilde{\boldsymbol{K}} = \left(\boldsymbol{I} - \frac{\mathbf{1}_{N\times N}}{N}\right)^T \boldsymbol{K} \left(\boldsymbol{I} - \frac{\mathbf{1}_{N\times N}}{N}\right)$$

where $\mathbf{1}_{N\times N}$ is an $N \times N$ matrix with all entries equal to 1.

5. Diagonalize $\tilde{\boldsymbol{K}}$ to get its eigenvalues $\boldsymbol{\Lambda} = diag(\lambda_1, \cdots, \lambda_n)$ and eigenvectors $\boldsymbol{A} = [\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_n]$.

6. Normalize the eigenvectors $\hat{\boldsymbol{A}} = [\hat{\boldsymbol{\alpha}}_i]_{i=1}^N = \left[\frac{\boldsymbol{\alpha}_i}{\sqrt{\lambda}}\right]_{i=1}^N$.

7. The c eigenvectors retained by `KPCA` correspond to c largest eigenvalues such that $\frac{\sum_{j=1}^{c}\lambda_j}{\sum_{j=1}^{N}\lambda_j}$ equals the desired variance to be captured.

8. Project the data points on the eigenvectors

$$\boldsymbol{y} = \hat{\boldsymbol{A}}^T \left(\boldsymbol{I} - \frac{\mathbf{1}_{N\times N}}{N}\right) \left(\begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}) \\ \cdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}) \end{bmatrix} - \boldsymbol{K}\frac{\mathbf{1}_{N\times 1}}{N} \right)$$

where $\mathbf{1}_{N\times 1}$ is an $N \times 1$ vector with all entries equal to 1.

9. Use the projections $\boldsymbol{y}$ instead of the data points.

5.1.2 Kernel Fisher Discriminant

As described in Section 3.2, the `FLDA` algorithm cannot cope with non-linearly separable classes; to overcome this limitation, in [66] the Kernel Fisher Discriminant (`KFD`) algorithm, is proposed exploiting the kernel trick to perform `FLDA` in a kernel space.

Consider a non invertible map $\phi : \boldsymbol{\mathcal{Q}}^D \rightarrow \boldsymbol{\mathcal{Q}}_\phi^h$ which maps points from the input euclidean space $\Re^D$ to an higher-dimensional inner-product space $\boldsymbol{\mathcal{Q}}_\phi^h$, the `FLDA` algorithm can be applied in $\boldsymbol{\mathcal{Q}}_\phi^h$ to the training points mapped through ϕ. Considering the two-class case, we call the following quantities respectively between and within-class scatter matrix in $\boldsymbol{\mathcal{Q}}_\phi^h$:

$$\boldsymbol{S}_{Bet}^\phi = (\boldsymbol{m}_b^\phi - \boldsymbol{m}_a^\phi)(\boldsymbol{m}_b^\phi - \boldsymbol{m}_a^\phi)^T \tag{5.1}$$

$$\begin{aligned} \boldsymbol{S}_W^\phi &= \sum_{n=1}^{N_A}(\phi(\boldsymbol{x}_n) - \boldsymbol{m}_a^\phi)(\phi(\boldsymbol{x}_n) - \boldsymbol{m}_a^\phi)^T \\ &\quad + \sum_{n=1}^{N_B}(\phi(\boldsymbol{x}_n) - \boldsymbol{m}_b^\phi)(\phi(\boldsymbol{x}_n) - \boldsymbol{m}_b^\phi)^T \end{aligned} \tag{5.2}$$

where $\boldsymbol{m}_a^\phi = \frac{1}{N_A}\sum_{i=1}^{N_A}\phi(\boldsymbol{x}_i^a)$, $\boldsymbol{m}_b^\phi = \frac{1}{N_B}\sum_{i=1}^{N_B}\phi(\boldsymbol{x}_i^b)$ are the means of class A and B in the space $\boldsymbol{\mathcal{Q}}_\phi^h$, N_A and N_B are the cardinality of class A and class B, and $N = N_A + N_B$.

The computation of the projection vector $\boldsymbol{w}$ by applying FLDA in $\boldsymbol{\mathcal{Q}}_\phi^h$ is equivalent to solving the following maximization problem:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{S}_{Bet}^\phi \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W^\phi \boldsymbol{w}} \tag{5.3}$$

To find the Fisher Subspace in $\boldsymbol{\mathcal{Q}}_\phi^h$, we need to reformulate the Equation (5.3) in terms of dot products between input points, so that they can be replaced by some kernel function. In [67] the authors show that any solution of $\boldsymbol{w} \in \boldsymbol{\mathcal{Q}}_\phi^h$ can be written as an expansion of the form:

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \phi(\boldsymbol{x}_i). \tag{5.4}$$

Using the definition of $\boldsymbol{m}_a^\phi$, $\boldsymbol{m}_b^\phi$, and the Equation (5.4) we obtain:

$$\boldsymbol{w}^T \boldsymbol{m}_a^\phi = \frac{1}{N_A}\sum_{i=1}^{N}\sum_{k=1}^{N_A} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}_k^a) = \boldsymbol{\alpha}^T \boldsymbol{M}_A. \tag{5.5}$$

$$\boldsymbol{w}^T \boldsymbol{m}_b^\phi = \frac{1}{N_B}\sum_{i=1}^{N}\sum_{k=1}^{N_B} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}_k^b) = \boldsymbol{\alpha}^T \boldsymbol{M}_B. \tag{5.6}$$

where $k(\cdot,\cdot)$ is the kernel function, and we have set:

$$(\boldsymbol{M}_A)_i = \frac{1}{N_A}\sum_{k=1}^{N_A} k(\boldsymbol{x}_i, \boldsymbol{x}_k^a) \quad (\boldsymbol{M}_B)_i = \frac{1}{N_B}\sum_{k=1}^{N_B} k(\boldsymbol{x}_i, \boldsymbol{x}_k^b)$$

replacing the dot products by the kernel function.

Considering the numerator of Equation (5.3), and using the definition of $\boldsymbol{S}_{Bet}^\phi$ and Equations (5.5,5.6), we can reformulate:

$$\boldsymbol{w}^T \boldsymbol{S}_{Bet}^\phi \boldsymbol{w} = \boldsymbol{\alpha}^T \boldsymbol{M} \boldsymbol{\alpha} \tag{5.7}$$

where $\boldsymbol{M} = (\boldsymbol{M_A} - \boldsymbol{M_B})(\boldsymbol{M_A} - \boldsymbol{M_B})^T$.
Similarly for the denominator:

$$\boldsymbol{w}^T \boldsymbol{S}_W^\phi \boldsymbol{w} = \boldsymbol{\alpha}^T \boldsymbol{N} \boldsymbol{\alpha} \tag{5.8}$$

where $\boldsymbol{N} = \boldsymbol{K}_a(\boldsymbol{I} - \boldsymbol{L}_a)\boldsymbol{K}_a^T + \boldsymbol{K}_b(\boldsymbol{I} - \boldsymbol{L}_b)\boldsymbol{K}_b^T$. Notice that $\boldsymbol{K}_a$ is the $N \times N_A$ matrix with $(\boldsymbol{K}_a)_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m^a)$, $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{L}_a$ is the matrix with all entries N_A^{-1}, and the corresponding quantities for class B are similarly evaluated.

Combining the Equations (5.7,5.8) we can find the Fisher subspace in $\boldsymbol{\mathcal{Q}}_\phi^h$ by maximizing:

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}^T \boldsymbol{M} \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \boldsymbol{N} \boldsymbol{\alpha}}$$

As explained in Section 3.2, the solution vector, $\boldsymbol{w}$, is the leading eigenvector of $\boldsymbol{N}^{-1}\boldsymbol{M}$, and the projection of a new point $\boldsymbol{x}$ onto $\boldsymbol{w}$ is performed by computing:

$$\boldsymbol{w} \cdot \phi(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}).$$

5.2 Kernel IPCAC

To relax the linear separability constraint, imposed by the IPCAC algorithm, it is possible to exploit the kernel trick as in the Kernel Principal Component Analysis, thus obtaining a Kernel Isotropic Principal Component Analysis Classifier (K-IPCAC).

The main idea is that the classes to be separated are non-linearly separable in the original space $\boldsymbol{\mathcal{Q}}^D \subseteq \Re^D$, but it is possible to map the N training points $\boldsymbol{p}_i \in \hat{\boldsymbol{\mathcal{P}}}$ in a higher dimensional space $\boldsymbol{\mathcal{Q}}_\phi^h$ where the classes are linearly separable; this is done through a non-invertible map $\boldsymbol{\phi}(\cdot)$, that is $\boldsymbol{\phi}(\boldsymbol{p}_i) \in \boldsymbol{\mathcal{Q}}_\phi^h$. The generated points are then used to compute the PCA in $\boldsymbol{\mathcal{Q}}_\phi^h$, thus obtaining a set of $\bar{N} \leq N$ relevant principal components $\{\boldsymbol{x}_k\}_{k=1}^{\bar{N}} \in \boldsymbol{\mathcal{Q}}_\phi^h$; the subspace spanned by the vectors $\{\boldsymbol{x}_k\}_{k=1}^{\bar{N}}$ is the KPCA subspace where the vectors $\boldsymbol{\phi}(\boldsymbol{p}_i)$ must be finally projected.

In [85] the authors prove that it is possible to compute a weight matrix $\boldsymbol{A} = \{\alpha_{ik}\}_{i,k=1}^{N,\bar{N}}$ that allows to calculate the projection of a point $\boldsymbol{\phi}(\boldsymbol{p})$ on the principal components $\{\boldsymbol{x}_k\}_{k=1}^{\bar{N}}$. To this aim, it is firstly assumed that the mapped points $\boldsymbol{\phi}(\boldsymbol{p}_i)$ are mean centered in $\boldsymbol{\mathcal{Q}}_\phi^h$; in this case the following steps must be performed:

1. compute the design (Gram) matrix $\boldsymbol{K} = \{K_{ik}\}_{i,k=1}^{N}$ with $K_{ik} = k(\boldsymbol{p}_i, \boldsymbol{p}_k)$, where the points $\{\boldsymbol{p}_i\}_{i=1}^{N}$ are training vectors, and $k(\boldsymbol{p}_i, \boldsymbol{p}_k)$

is the kernel function that allows to compute the dot product of $\boldsymbol{\phi}(\boldsymbol{p}_i)$ and $\boldsymbol{\phi}(\boldsymbol{p}_k)$;

2. compute the eigen-decomposition $\boldsymbol{K} = \bar{\boldsymbol{A}}\bar{\boldsymbol{\Lambda}}\bar{\boldsymbol{A}}^T$, and remove eventual zero-variance components obtaining the new decomposition $\boldsymbol{K} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{\Lambda}}\tilde{\boldsymbol{A}}^T$, where $\bar{N}$ components are retained;

3. compute the weight matrix $\boldsymbol{A} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}}$.

Having computed $\boldsymbol{A}$, the generic point $\boldsymbol{\phi}(\boldsymbol{p}) \in \boldsymbol{\mathcal{Q}}_{\phi}^{h}$ can be projected on $\{\boldsymbol{x}_k\}_{k=1}^{\bar{N}}$ as:

$$\boldsymbol{x}_k \cdot \boldsymbol{\phi}(\boldsymbol{p}) = \sum_{i=1}^{N} \alpha_{ik} \boldsymbol{\phi}(\boldsymbol{p}_i) \cdot \boldsymbol{\phi}(\boldsymbol{p}) = \sum_{i=1}^{N} \alpha_{ik} k(\boldsymbol{p}_i, \boldsymbol{p}) \tag{5.9}$$

When the training points $\boldsymbol{\phi}(\boldsymbol{p}_i)$ are not mean centered in $\boldsymbol{\mathcal{Q}}_{\phi}^{h}$, the algorithm described above cannot be directly applied; therefore, as shown in [85], to calculate $\boldsymbol{A}$ the mean centered matrix $\tilde{\boldsymbol{K}}$ must be employed instead of $\boldsymbol{K}$. $\tilde{\boldsymbol{K}}$ is obtained as follows:

$$\tilde{\boldsymbol{K}} = \boldsymbol{K} - \boldsymbol{1}_N\boldsymbol{K} - \boldsymbol{K}\boldsymbol{1}_N + \boldsymbol{1}_N\boldsymbol{K}\boldsymbol{1}_N \tag{5.10}$$

where $\boldsymbol{1}_N = \{N^{-1}\}_{i,k=1}^{N}$.

Exploiting these theoretical results, we derived a method to compute the Fisher subspace on training data projected on the KPCA subspace. To describe our method we start by considering a training set mean centered in $\boldsymbol{\mathcal{Q}}_{\phi}^{h}$, and noting that Equation (5.9) can be restated in matrix form as:

$$\{\boldsymbol{x}_k \cdot \boldsymbol{\phi}(\boldsymbol{p})\}_{k=1}^{\bar{N}} = \boldsymbol{A}^T\{k(\boldsymbol{p}_i, \boldsymbol{p})\}_{i=1}^{N} = \boldsymbol{A}^T\boldsymbol{K}(\boldsymbol{p})$$

The first step of our method obtains the isotropic components of each training point $\boldsymbol{p_i}$, by using as scaling factor the inverse square root of the diagonal elements λ_k of $\boldsymbol{\Lambda}$ (where $\boldsymbol{\Lambda} = \tilde{\boldsymbol{\Lambda}}N^{-1}$ as shown in [85]), that is:

$$\{\lambda_k^{-\frac{1}{2}}\boldsymbol{x}_k \cdot \boldsymbol{\phi}(\boldsymbol{p})\}_{k=1}^{\bar{N}} = \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{A}^T\boldsymbol{K}(\boldsymbol{p}) \tag{5.11}$$

Next, we represent the projection of the training points $\hat{\boldsymbol{\mathcal{P}}} \subset \boldsymbol{\mathcal{Q}}^D$ on the principal components $\{\boldsymbol{x}_k\}_{k=1}^{\bar{N}} \subset \boldsymbol{\mathcal{Q}}_\phi^h$ with a matrix $\boldsymbol{P}_\phi$ obtained as follows:

$$\begin{aligned}
\boldsymbol{P}_\phi &= \{\lambda_k^{-\frac{1}{2}} \boldsymbol{x}_k \cdot \boldsymbol{\phi}(\boldsymbol{p}_i)\}_{i,k=1}^{N,\bar{N}} \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{A}^T \boldsymbol{K} \\
&= (\tilde{\boldsymbol{\Lambda}} N^{-1})^{-\frac{1}{2}} (\tilde{\boldsymbol{A}} \tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}})^T (\tilde{\boldsymbol{A}} \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{A}}^T) \\
&= N^{\frac{1}{2}} \tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}} \tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{A}} \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{A}}^T = N^{\frac{1}{2}} \tilde{\boldsymbol{A}}^T
\end{aligned} \tag{5.12}$$

being $\tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{A}} = \boldsymbol{I}$ for the orthogonality of $\tilde{\boldsymbol{A}}$.

Finally, the Fisher subspace is calculated by employing the cluster means of the points represented by the columns of $\boldsymbol{P}_\phi$.

To relax the hypothesis about mean centering of the training points in $\boldsymbol{\mathcal{Q}}_\phi^h$, we consider the projections of the centered points $\{\boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi\}_{i=1}^N$, and we exploit the result reported in Equation (5.10). More precisely, calling $\boldsymbol{\mu}_\phi = N^{-1} \sum_i \boldsymbol{\phi}(\boldsymbol{p}_i)$ the mean of the training points mapped in $\boldsymbol{\mathcal{Q}}_\phi$, we compute the matrix $\boldsymbol{P}_\phi$ as follows:

$$\begin{aligned}
\boldsymbol{P}_\phi &= \left\{ \lambda_k^{-\frac{1}{2}} \boldsymbol{x}_k \cdot (\boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi) \right\}_{i,k=1}^{N,\bar{N}} \\
&= \left\{ \sum_j \lambda_k^{-\frac{1}{2}} \alpha_{ik} (\boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi) \cdot (\boldsymbol{\phi}(\boldsymbol{p}_j) - \boldsymbol{\mu}_\phi) \right\}_{i,k=1}^{N,\bar{N}} \\
&= \left\{ \sum_j \lambda_k^{-\frac{1}{2}} \alpha_{ik} \left(\boldsymbol{\phi}(\boldsymbol{p}_i) \cdot \boldsymbol{\phi}(\boldsymbol{p}_j)\right) \right\}_{i,k=1}^{N,\bar{N}} \\
&\quad - \left\{ \sum_j \lambda_k^{-\frac{1}{2}} \alpha_{ik} \left(\boldsymbol{\phi}(\boldsymbol{p}_i) \cdot \boldsymbol{\mu}_\phi\right) \right\}_{i,k=1}^{N,\bar{N}} \\
&\quad - \left\{ \sum_j \lambda_k^{-\frac{1}{2}} \alpha_{ik} \left(\boldsymbol{\mu}_\phi \cdot \boldsymbol{\phi}(\boldsymbol{p}_j)\right) \right\}_{i,k=1}^{N,\bar{N}} \\
&\quad + \left\{ \sum_j \lambda_k^{-\frac{1}{2}} \alpha_{ik} \left(\boldsymbol{\mu}_\phi \cdot \boldsymbol{\mu}_\phi\right) \right\}_{i,k=1}^{N,\bar{N}} \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{A}^T (\boldsymbol{K} - \boldsymbol{1}_N \boldsymbol{K} - \boldsymbol{K} \boldsymbol{1}_N + \boldsymbol{1}_N \boldsymbol{K} \boldsymbol{1}_N) \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{A}^T \tilde{\boldsymbol{K}} = N^{\frac{1}{2}} \tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}} \tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{A}} \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{A}}^T = N^{\frac{1}{2}} \tilde{\boldsymbol{A}}^T
\end{aligned} \tag{5.13}$$

Note that $\tilde{\boldsymbol{K}} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{\Lambda}}\tilde{\boldsymbol{A}}^T \neq \boldsymbol{K}$, and the result reported in Equation (5.13) is the same as that reported in Equation (5.12) .

Assuming that the first N_A column vectors $\boldsymbol{P}_{\phi|1..N_A}$ of $\boldsymbol{P}_\phi$ belong to class A, and that the remaining column vectors $\boldsymbol{P}_{\phi|N_A+1..N}$ belong to class B, and noting that these points are whitened through the KPCA algorithm, it is possible to use them for a direct Fisher subspace estimation. To this aim, we must compute the quantities:

$$\begin{aligned}\boldsymbol{\mu}_{\boldsymbol{A}\boldsymbol{\phi}} &= \langle \boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi \rangle_{i=1}^{N_A} \\ \boldsymbol{\mu}_{\boldsymbol{B}\boldsymbol{\phi}} &= \langle \boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi \rangle_{i=N_A+1}^{N}\end{aligned}$$

and their difference:

$$\begin{aligned}\bar{\boldsymbol{F}} &= \langle \boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi \rangle_{i=1}^{N_A} - \langle \boldsymbol{\phi}(\boldsymbol{p}_i) - \boldsymbol{\mu}_\phi \rangle_{i=N_A+1}^{N} \\ &= \boldsymbol{P}_{\phi|1..N_A} \left(\underbrace{N_A^{-1} \cdots}_{N_A \text{ times}} \quad \underbrace{0 \cdots}_{N_B \text{ times}} \right)^T \\ &\quad -\boldsymbol{P}_{\phi|N_A+1..N} \left(\underbrace{0 \cdots}_{N_A \text{ times}} \quad \underbrace{N_B^{-1} \cdots}_{N_B \text{ times}} \right)^T \\ &= \boldsymbol{P}_\phi \left(\underbrace{N_A^{-1} \cdots}_{N_A \text{ times}} \quad \underbrace{-N_B^{-1} \cdots}_{N_B \text{ times}} \right)^T = N^{\frac{1}{2}} \tilde{\boldsymbol{A}}^T \boldsymbol{N}_{A|B}^{-1}\end{aligned}$$

where we have defined $\boldsymbol{N}_{A|B}^{-1} = \left(\underbrace{N_A^{-1} \cdots}_{N_A \text{ times}} \quad \underbrace{-N_B^{-1} \cdots}_{N_B \text{ times}} \right)^T$.

The vector $\bar{\boldsymbol{F}}$ must be normalized; its norm is:

$$\begin{aligned}\|\bar{\boldsymbol{F}}\| &= \left\| N^{\frac{1}{2}} \tilde{\boldsymbol{A}}^T \boldsymbol{N}_{A|B}^{-1} \right\| = N^{\frac{1}{2}} \left\| \boldsymbol{N}_{A|B}^{-1} \right\| \\ &= N^{\frac{1}{2}} \sqrt{N_A \left(N_A^{-2}\right) + N_B \left(N_B^{-2}\right)} = N (N_A N_B)^{-\frac{1}{2}}\end{aligned}$$

thus, the Fisher subspace can be computed as follows:

$$\boldsymbol{F} = \frac{\bar{\boldsymbol{F}}}{\|\bar{\boldsymbol{F}}\|} = N^{-\frac{1}{2}} (N_A N_B)^{\frac{1}{2}} \tilde{\boldsymbol{A}}^T \boldsymbol{N}_{A|B}^{-1} \tag{5.14}$$

In particular, if $N_A = N_B$ we get:

$$\boldsymbol{F} = \frac{\bar{\boldsymbol{F}}}{\|\bar{\boldsymbol{F}}\|} = N^{-\frac{1}{2}} \tilde{\boldsymbol{A}}^T \left(\underbrace{1 \cdots}_{N_A \text{ times}} \quad \underbrace{-1 \cdots}_{N_B \text{ times}} \right)^T$$

Given a testing point $\boldsymbol{p}$, we must compute its projection on $\boldsymbol{F}$; to this aim we use Equation (5.11) and Equation (5.14):

$$\begin{aligned} proj_{\boldsymbol{F}}(\boldsymbol{p}) &= \left(\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{A}^T\boldsymbol{K}(\boldsymbol{p})\right)^T(\boldsymbol{F}) \\ &= \left(\boldsymbol{K}(\boldsymbol{p})^T\tilde{\boldsymbol{A}}\tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}}\tilde{\boldsymbol{\Lambda}}^{-\frac{1}{2}}N^{\frac{1}{2}}\right)\left(N^{-\frac{1}{2}}(N_AN_B)^{\frac{1}{2}}\tilde{\boldsymbol{A}}^T\boldsymbol{N}_{A|B}^{-1}\right) \\ &= \boldsymbol{K}(\boldsymbol{p})^T\underbrace{\left((N_AN_B)^{\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{\Lambda}}^{-1}\tilde{\boldsymbol{A}}^T\boldsymbol{N}_{A|B}^{-1}\right)}_{\boldsymbol{w}} \\ &= \boldsymbol{K}(\boldsymbol{p})^T\boldsymbol{w} \end{aligned}$$

Note that, since the weight vector $\boldsymbol{w}$ can be precomputed at training time, the classification algorithm is similar to that obtained by Equation (3.16), that is $\boldsymbol{w}\cdot\boldsymbol{K}(\boldsymbol{p}) > \gamma$, where the thresholding value γ is estimated using the same algorithm proposed for IPCAC through Equation (3.17).

The obtained classifier requires, for each testing point, only N kernel function evaluations more than the IPCAC algorithm, thus remaining very simple and efficient.

5.2.1 K-IPCAC Retaining Variance

In [102] the authors analyze the kernel Fisher discriminant method, demonstrating that it is equivalent to KPCA plus Fisher linear discriminant analysis. Based on this result, they proposed a different KFD algorithm based on a first step of KPCA followed by LDA for a second feature extraction in the KPCA-transformed space. K-IPCAC offers a similar representation, guaranteeing a methodology that reduces the time and space requirements.

Furthermore, to reduce the problems of overfitting that affect the kernel methods it is possible to retain only a percentage of the variance expressed by the eigenvalues (and the correlated eigenvectors) calculated during the diagonalization of the Gram-matrix [1] (see step 2 in Section 5.2), thus performing a kind of dimensionality reduction in the inner-product space and maintaining only the largest part of the spectrum that is less affected by the noise. This approach (called K-IPCAC$_{var}$) allows to achieve better results

[1] This step is performed by retaining the d eigenvectors corresponding to the d largest eigenvalues such that $\frac{\sum_{j=1}^{d}\lambda_j}{\sum_{j=1}^{N}\lambda_j}$ equals the desired variance to be captured.

during the classification task, as it is confirmed by the experimental results reported in Section 5.3.2.

5.2.2 Kernel Truncated IPCAC

In Section 5.2 we prove that a given point $\boldsymbol{p}$ can be projected on Fs in the kernel space as follows:

$$proj_{\boldsymbol{F}}(\boldsymbol{p}) = \boldsymbol{K}(\boldsymbol{p})^T \underbrace{\left((N_A N_B)^{\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{\Lambda}}^{-1} \tilde{\boldsymbol{A}}^T \boldsymbol{N}_{A|B}^{-1}\right)}_{\boldsymbol{w}} = \boldsymbol{K}(\boldsymbol{p})^T \boldsymbol{w}$$

where $\boldsymbol{K}(\boldsymbol{p}) = \{KerFunction(\boldsymbol{p}_i, \boldsymbol{p})\}_{i=1}^{N}$ is the vector of the kernel values computed between the point $\boldsymbol{p}$ and the set of the training points $\boldsymbol{p}_i$, N is the cardinality of the training set, N_A and N_B are the cardinalities of the two classes, $\tilde{\boldsymbol{\Lambda}}$ are the eigenvalues obtained by the decomposition of the kernel matrix, $\tilde{\boldsymbol{A}}$ are the associated eigenvectors, and $\boldsymbol{N}_{A|B}^{-1} = \left(\underbrace{N_A^{-1} \cdots}_{N_A \text{ times}} \quad \underbrace{-N_B^{-1} \cdots}_{N_B \text{ times}}\right)^T$. We extend this method by exploiting the same concept at the basis of the algorithm presented in Section 4.3, where we consider N points in $\Re^D$ ($N/D \approx \alpha \approx 1$) and we avoid any dimensionality reduction technique, by applying a method to whiten the data in the linear subspace $\pi_d = \mathbf{Span}\langle \boldsymbol{v}_1, \cdots, \boldsymbol{v}_d \rangle$, spanned by the first $d \ll D$ principal components, while maintaining unaltered the information related to the orthogonal subspace $(\pi_d)^{\perp} = \mathbf{Span}\langle \boldsymbol{v}_{d+1}, \cdots, \boldsymbol{v}_D \rangle$. More precisely, in this case we select the largest eigenvalues that represent a fixed amount of variance defined a-priori (as in Section 5.2.1), and we set the remaining part of the spectrum to 1; this process reduces the overfitting problems produced by the smallest part of the spectrum without loosing any kind of information. The advantages of employing this modified version of K-IPCAC, called K-TIPCAC, are supported by our experimental results reported in Table 5.3.

5.3 Experimental Results

In this section we at first evaluate the performance of K-IPCAC by applying it to the task of Spam message classification (see Section 5.3.1); this experiment allows to compare the achieved results to those obtained by well known classifiers. Secondly, in Section 5.3.2, we employ a synthetic dataset

to compare the performances of K-IPCAC, K-TIPCAC, and K-IPCAC$_{var}$ when the cardinality of the training set is approximately equal to the space dimensionality, and the data are drawn from two MGDs.

5.3.1 Experiments on the Spam Classification

To evaluate the effectiveness of K-IPCAC[2] we test it on the Spam classification problem described in Section 3.7, employing the same dataset and performing the same preprocessing steps[3].

Experiment$_{\#}$	*Classifier*	*Accuracy (std)*	*Precision*	*Recall*
Exp$_1$	KNN	95.499 (0.759)	95.922	95.028
	IPCAC$_{var}$	96.817 (0.279)	96.250	97.430
	SVM	97.116 (0.213)	96.234	98.057
	K-IPCAC	***97.566*** (0.139)	***96.535***	***98.548***
Exp$_2$	KNN	95.449	95.113	95.927
	IPCAC$_{var}$	95.05	93.901	96.374
	SVM	97.001	95.958	98.133
	K-IPCAC	96.917	95.447	98.533
	MM-IPCAC	**98.350**	**97.387**	**99.367**

Table 5.1: Experimental results on the emails belonging to TREC corpus.

Experiment$_{\#}$	*Classifier*	*Accuracy (std)*	*Precision*	*Recall*
Exp$_1$	KNN	96.278 (0.379)	96.773	95.777
	IPCAC$_{var}$	97.444 (1.002)	97.608	97.333
	SVM	98.444 (0.602)	98.148	98.778
	K-IPCAC	**98.889** (0.181)	**98.569**	**99.222**
Exp$_2$	KNN	91.222	96.932	85.111
	IPCAC$_{var}$	90.167	93.280	86.667
	SVM	93.611	96.444	90.556
	K-IPCAC	93.222	94.209	92.111
	MM-IPCAC	***98.222***	***98.329***	***98.111***

Table 5.2: Experimental results on the emails belonging to SpamAssassin corpus.

Results reported in Tables 5.1 and 5.2 are commented below:

Experiment 1: on both corpuses K-IPCAC outperforms the other classifiers, thus proving that it is the best performing classifier when a single

[2]In these experiments we employ the base algorithm, with a RBF kernel suitably tuned.
[3]All the tested algorithms are implemented in Python as described in Section 5.3.1.

training set is available.

Experiment 2: on both corpuses the results show that our MM-IPCAC technique is promising, since it outperforms all the other algorithms.

Note that, on the TREC corpus (see Table 5.1), the best accuracy over both the experiments is achieved by the MM-IPCAC algorithm; instead, on the SpamAssassin corpus (see Table 5.2) K-IPCAC obtains the best results, while MM-IPCAC obtains anyway a good performance. Moreover, when Experiment 1 is run on both the corpuses, K-IPCAC seems to be the most reliable classifier since it achieves the smallest standard deviation of the accuracy parameter.

These experiments confirm the effectiveness of the proposed algorithm, that overcomes the SVM classifier in all the performed tests.

5.3.2 Experimental Comparison between K-IPCAC, K-TIPCAC and K-IPCAC$_{var}$

To compare the algorithms proposed in Section 5.2 we have generated three synthetic datasets in $\Re^{500}$. Each dataset is composed by two classes whose points were drawn from 4 MGDs per class. Each MGD is an isotropic Gaussian with standard deviation $e = 1$.

The MGD means of the first class were set to:

- $(0, \ldots, 0)^T$: a vector with 500 zeros;
- $(1, \ldots, 1, 0, \ldots, 0)^T$: a vector with 250 ones and 250 zeros;
- $(0.5, \ldots, 0.5, -1, \ldots, -1)^T$: a vector with 250 values fixed to 0.5 and 250 values fixed to -1;
- $(0.5, \ldots, 0.5, 1, \ldots, 1)^T$: a vector with 250 values fixed to 0.5 and 250 ones.

The means of the MGDs used for the second class were:

- $(1, \ldots, 1)^T$: a vector with 500 ones;
- $(1, \ldots, 1, -1, \ldots, -1)^T$: a vector with 250 ones and 250 values fixed to -1;

- $(0, \ldots, 0, 1, \ldots, 1)^T$: a vector with 250 zeros and 250 ones;
- $(0, \ldots, 0, -1, \ldots, -1)^T$: a vector with 250 zeros and 250 values fixed to -1.

All the three datasets are balanced and they differ for the cardinality of the training set. More precisely, **Dataset1** is composed by 1000 points per class (250 points are drawn from each `MGD`), **Dataset2** is composed by 500 points per class (125 points are drawn from each `MGD`), and **Dataset3** is composed by 200 points per class (50 points are drawn from each `MGD`).

These datasets were employed to compare the results computed by `K-IPCAC`, `K-IPCAC`$_{var}$, and `K-TIPCAC`[4], when performing 100 runs of 10-fold cross validation and averaging the obtained results. The adopted evaluation measures are the *Accuracy*, *Specificity*, and *Sensitivity* described in Appendix A. All the described experiments were executed by using a MATLAB implementation of the tested algorithms.

Dataset	*Classifier*	*Accuracy (std)*	*Specificity*	*Sensitivity*
1	`K-IPCAC`	95.92 (0.738)	94.74	97.10
	`K-IPCAC`$_{var}$	96.01 (0.882)	95.01	97.01
	`K-TIPCAC`	**96.25 (0.949)**	**95.40**	**97.10**
	`SVM`	95.96 (0.857)	97.01	94.90
2	`K-IPCAC`	84.88 (2.424)	84.57	85.19
	`K-IPCAC`$_{var}$	85.20 (2.898)	85.20	85.20
	`K-TIPCAC`	**85.83 (2.359)**	**85.64**	**86.02**
	`SVM`	84.37 (2.542)	84.82	83.92
3	`K-IPCAC`	79.02 (7.284)	78.00	80.04
	`K-IPCAC`$_{var}$	79.48 (6.646)	81.46	77.50
	`K-TIPCAC`	**80.29 (4.158)**	**81.53**	**79.05**
	`SVM`	78.75 (6.872)	79.60	77.90

Table 5.3: Experimental results obtained on the datasets generated with $e = 1$.

As shown in Table 5.3 the `K-TIPCAC` obtains the best results and `K-IPCAC`$_{var}$ outperforms the base algorithm `K-IPCAC` and `SVM`; this confirms the promising quality of the proposed modification. Furthermore, it is important to

[4]In these experiments for `K-TIPCAC`, and `K-IPCAC`$_{var}$ we retained a percentage of variance equal to 99%.

underline that reducing the number of points drawn by each MGD, the gap between the performance achieved by K-TIPCAC and K-IPCAC increases; this fact shows that this algorithm has a stronger generalization capability.

5.4 Perceptron-IPCAC

To overcome the limitations of linear classifiers while maintaining a low computational complexity, we developed a new binary classification algorithm, Perceptron-IPCAC, that is based on the assumption that the pdfs underlying each class are MoGs. More precisely, P-IPCAC has been designed as a multilayer perceptron that combines the classification results of a set of linear classifiers. In the following the two classes, whose class conditional distributions are MoGs, will be referred as class A and B; the training points belonging to these classes are $\boldsymbol{\mathcal{P}}_A = \{\boldsymbol{p}_{A,j}\}_{j=1}^{N_A} \subset \Re^D$ and $\boldsymbol{\mathcal{P}}_B = \{\boldsymbol{p}_{B,j}\}_{j=1}^{N_B} \subset \Re^D$.

In this section we firstly describe a clustering step that estimates the parameters of each MoG (see Section 5.4.1), secondly we describe the P-IPCAC method (see Section 5.4.2).

5.4.1 Estimation of MoG Parameters

Given the set of labeled training points $\boldsymbol{\mathcal{P}} = \boldsymbol{\mathcal{P}}_A \bigcup \boldsymbol{\mathcal{P}}_B$ (with $|\boldsymbol{\mathcal{P}}_A| = N_A$ and $|\boldsymbol{\mathcal{P}}_B| = N_B$), we assume that each class is distributed according to a MoG and we apply a preprocessing step for estimating its parameters.

For the sake of robustness we focus on the definition of a parameterless algorithm that processes the set of training points belonging to one class as follows:

1. the X-Means method [28] is applied to cluster the training points in the considered class, and the resulting number of clusters is considered as the number of MGDs underlying the class (see Figure 5.2);

2. the EM algorithm is employed to estimate the parameters of each MoG (see Figure 5.3);

3. a Gaussian merging procedure (see the next Subsection) is applied to merge those estimated distributions that are strongly overlapped, thus obtaining the final set of MGDs underlying the class (see Figure 5.4);

4. the estimated MGDs are used to cluster the training data belonging to the considered class.

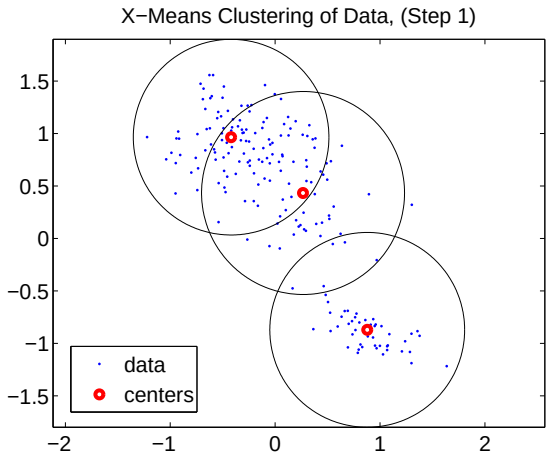


Figure 5.2: Result after Application of X-Means step on 2D data.

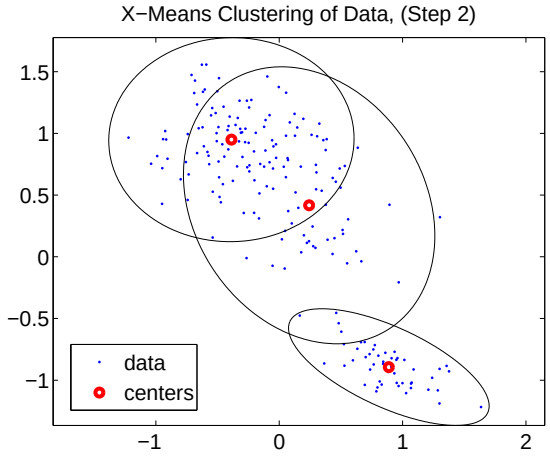


Figure 5.3: Result after Application of the Expectation Maximization step on 2D data.

We note that, using the estimated clusters to train P-IPCAC, the achieved classification results are greater that those computed by using only the clusters extracted by step 1. Nevertheless, to reduce the computational complexity of the preprocessing step, it is possible to apply only X-Means during this phase.

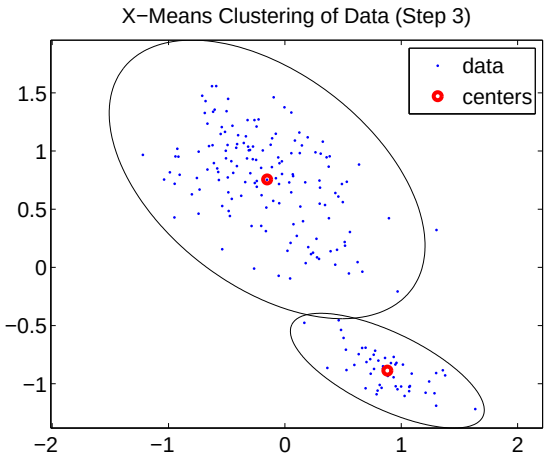


Figure 5.4: Result after Application of the Merging procedure on 2D data.

We further note that, it would be possible to employ directly the MGDs to perform classification, but the obtained results would be strongly depending on the quality of the pdfs estimated on the training data, and the classification time would be quadratic in the space dimensionality.

Gaussian merging procedure

By employing the mean and the covariance estimated during the step 2 described in Section 5.4.1 we consider the following criterion:

Merge criterion 1. *Two gaussians are merged if and only if the mean of one of them is inside the* 99.7% *of the confidence interval with respect to the other one. More precisely, consider the probability value* $\hat{p}$ *such that* $\int_{\Re^D} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \boldsymbol{\mathcal{X}}_{\mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i)\geq\hat{p}} d\boldsymbol{x} = 0.997$*, where* $\mathcal{N}(\cdot; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ *is the* i*-th gaussian function,* $\boldsymbol{\mathcal{X}}$ *is the indicator function, and the value* 0.997 *is referred to the confidence interval; the* j*-th gaussian is merged with the* i*-th one if* $\mathcal{N}(\boldsymbol{\mu}_j; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \geq \hat{p}$.

On the basis of the above mentioned criterion, the merging procedure proceeds as follows:

1. for each couple of clusters do:
    - if the Merge criterion is valid, merge the two clusters and estimate the new mean and the new covariance matrix;

- otherwise keep the couple of clusters unchanged;

2. repeat step 1 until there are not couples of clusters whereby the Merge criterion is valid.

5.4.2 P-IPCAC

In this section we describe the Perceptron-IPCAC, that is trained on the clustered classes $A = \bigcup_{n=1}^{M_A} A_n$ and $B = \bigcup_{n=1}^{M_B} B_n$, where M_A and M_B are the numbers of clusters contained in class A and in class B, respectively.

P-IPCAC has been developed exploiting the idea employed by the Decision Directed Acyclic Graph (DDAG) algorithm [78] for the multiclass classification. More precisely, $M_A M_B$ (IPCAC) binary classifiers are initially trained to perform the comparisons between the training points drawn from each Gaussian of the first class, with those drawn from each Gaussian of the second one (we recall that these training points are those belonging to the subclasses $\{A_n\}_{n=1...M_A}$ and $\{B_n\}_{n=1...M_B}$), in a one-against-one fashion.

For each binary classifier, the classification of a new point $\boldsymbol{p}$ is accomplished by computing:

$$\text{sign}\left(\boldsymbol{w}^T\boldsymbol{p} - \gamma\right) \tag{5.15}$$

where $\boldsymbol{w}$ is a weight vector, γ is a thresholding value, and sign $(\cdot)$ is the sign function.

We denote with $\boldsymbol{w}_{ij}$ and γ_{ij} the parameters related to the classifier trained to distinguish between points belonging to the subclasses A_i (i-th Gaussian) and B_j (j-th Gaussian); moreover, we define the weight matrix and the threshold vector as follows:

$$\begin{aligned} \boldsymbol{W} &= \{\boldsymbol{w}_{ij}\}_{i=1..M_A, j=1..M_B} \\ \boldsymbol{\gamma} &= \{\gamma_{ij}\}_{i=1..M_A, j=1..M_B} \end{aligned}$$

Using this notation, the responses of the $M_A M_B$ trained classifiers become:

$$\boldsymbol{z}(\boldsymbol{p}) = \{z_{ij}\}_{i=1..M_A, j=1..M_B} = \text{sign}\left(\boldsymbol{W}^T\boldsymbol{p} - \boldsymbol{\gamma}\right) \tag{5.16}$$

where $z_{ij} = -1$ when $\boldsymbol{p}$ is classified as belonging to subclass A_i, and $z_{ij} = 1$ when $\boldsymbol{p}$ is classified as belonging to subclass B_j.

Note that the evaluation of the $M_A M_B$ linear classifiers corresponds to the computation of a feed-forward neural network with D neurons for the input layer, representing the D features of the input vector $\boldsymbol{p}$, and $M_A M_B$ neurons for the first hidden layer (see Figure 5.5). More precisely, the synaptic weight for the hidden neuron with indexes (i, j) is $\boldsymbol{w}_{ij}$, whilst its activation threshold is γ_{ij}, and its activation function is $\operatorname{sign}(\cdot)$.

Using the game theory metaphor, the two MoGs can be seen as two teams, the linear classifiers in the first hidden layer of the network can be seen as the set of $M_A M_B$ matches comparing every pair of adversary members taken from the two teams, and the activation signals stored in $\boldsymbol{z}$ represent the results of all these matches.

Notice that the network structure described above does not highlight the behavior of each team member. To avoid the loss of this information a new neuron layer is required. This second hidden layer contains one neuron per team member, and each neuron collects its results through the matches where it participated. More precisely, denoting with $x_{A,n}$ the neuron associated to the n-th member of class A, it computes the following value:

$$x_{A,n} = \frac{1}{M_B} \sum_{j=1}^{M_B} z_{nj}$$

The results obtained for the members of the team A are negative when the member collects more victories than losses, and positive otherwise. On the other hand, the neuron $x_{B,n}$ associated to the n-th member of class B computes the following value:

$$x_{B,n} = \frac{1}{M_A} \sum_{i=1}^{M_A} z_{in}$$

The results obtained for the members of the team B are positive when the member collects more victories than losses, and negative otherwise.

The activation signals for the n-th members of the two teams are com-

puted as follows:

$$\begin{aligned} y_{A,n} &= \phi(-x_{A,n}) = \phi\left(-\frac{1}{M_B}\sum_{j=1}^{M_B} z_{nj}\right) \\ y_{B,n} &= \phi(x_{B,n}) = \phi\left(\frac{1}{M_A}\sum_{i=1}^{M_A} z_{in}\right) \end{aligned}$$

where $\phi(\cdot)$ is the activation function. To define it we consider the following particular cases:

$x_{B,n} \leq 0$: as mentioned above, in this case the n-th member of class B collects more losses than victories during the classification of point $\boldsymbol{p}$. Therefore we set $\phi(x_{B,n}) = 0$.

$x_{B,n} = 1 \quad \wedge \quad x_{B,i\neq n} = -1$: in this case, only the n-th member wins every match, while all the other members of class B lose every match. Under these circumstances we wish to classify the point $\boldsymbol{p}$ as belonging to class B; to this aim, it must be:

$$\sum_{n=1}^{M_B} \phi(x_{B,n}) > \sum_{n=1}^{M_A} \phi(-x_{A,n}) \tag{5.17}$$

In this case we consider that: being $\phi(x_{B,i\neq n}) = 0$, as we described for the case $x_{B,n} \leq 0$, it is $\sum_{n=1}^{M_B} \phi(x_{n_B}) = \phi(1) = 1$; moreover, since each member of the team A won $M_B - 1$ matches, gaining $M_B - 1$ points and losing only one point, $-x_{A,n} = \frac{M_B-2}{M_B}$.

Choosing as activation function $\phi(x) = x^m$, we can re-write Equation (5.17) as $1 > M_A \left(\frac{M_B-2}{M_B}\right)^m$. This equation is satisfied for the following values of m:

$$m > \frac{1}{\log_{M_A} \frac{M_B}{M_B-2}} \tag{5.18}$$

for $M_B \neq 2$ and $M_A \neq 1$. For this reason setting:

$$\begin{aligned} M'_B &= \max(M_B, 3) \\ M'_A &= \max(M_A, 3) \end{aligned}$$

m can be set to the following value:

$$m = \left\lceil \frac{1}{\log_{M'_A} \frac{M'_B}{M'_B - 2}} \right\rceil + 1 \tag{5.19}$$

Analogously, considering the particular case:

$$-x_{A,n} = 1 \wedge -x_{A,i \neq n} = -1$$

we can derive m' as:

$$m' = \left\lceil \frac{1}{\log_{M'_B} \frac{M'_A}{M'_A - 2}} \right\rceil + 1 \tag{5.20}$$

Summarizing, at training time we set the activation function $\phi(x) = x^t$, where t is fixed as follows:

$$t = \max \left(\left\lceil \frac{1}{\log_{M'_A} \frac{M'_B}{M'_B - 2}} \right\rceil, \left\lceil \frac{1}{\log_{M'_B} \frac{M'_A}{M'_A - 2}} \right\rceil \right) + 1 \tag{5.21}$$

The results related to all the members of each team must be processed by a "jury" to determine the winning team, and producing the classification result. The jury is represented by the output neuron. This last neuron has synaptic weights fixed to 1, and sign $(\cdot)$ as its activation function.

Summarizing, the following equation represents the network:

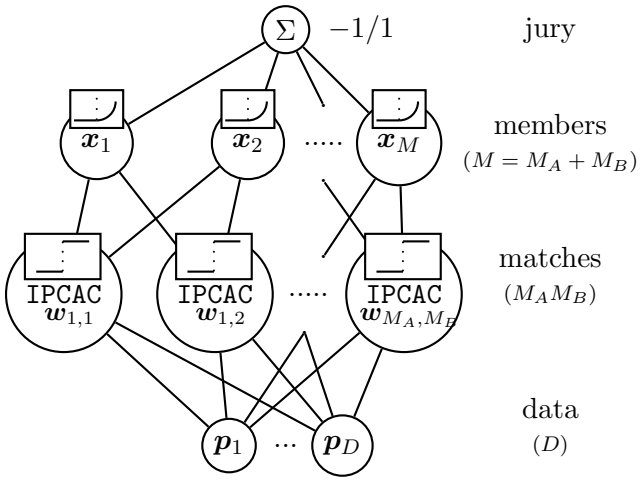
$$\begin{aligned} c(\boldsymbol{p}) \quad = \quad & \operatorname{sign} \left(-\sum_{n=1}^{M_A} \phi \left(-\frac{1}{M_B} \sum_{j=1}^{M_B} \operatorname{sign} \left(\boldsymbol{w}_{nj}^T \boldsymbol{p} - \gamma_{nj} \right) \right) + \right. \\ & \left. \sum_{n=1}^{M_B} \phi \left(\frac{1}{M_A} \sum_{i=1}^{M_A} \operatorname{sign} \left(\boldsymbol{w}_{in}^T \boldsymbol{p} - \gamma_{in} \right) \right) \right) \end{aligned} \tag{5.22}$$

where $c(\boldsymbol{p})$ is the classification of a new point $\boldsymbol{p}$. This quantity takes the value -1 if $\boldsymbol{p}$ is assigned to class A, and $+1$ if it is assigned to class B.

Note that the same behavior obtained by this network can be represented by the following expression:

$$\operatorname{sign} \left(\max_{n=1}^{M_B} (x_{B,n}) - \max_{n=1}^{M_A} (-x_{A,n}) \right) \tag{5.23}$$

Figure 5.5: *The P-IPCAC classifier structure is a three-layer perceptron, its behavior can be interpreted as that of two teams containing M^A and M^B members, matching, collecting results and being evaluated by a jury.*



5.4.3 Computational Complexity

In this section we derive the P-IPCAC computational complexity both for the training and for the classification phase.

The computational complexity of the P-IPCAC's training phase is dominated by the evaluation of the weights in the first hidden layer of the network; this step requires the training of $M_A M_B$ linear classifiers. Choosing IPCAC as the linear classifier adopted for the nodes in this layer, the training of each linear classifier requires to compute the eigenvalues and the eigenvectors of the covariance matrix estimated from the given data. The most efficient way to derive these informations is the computation of the Singular Value Decomposition (SVD) of the samples matrix. Representing with N the number of samples, and with D the space dimensionality, SVD requires at most $O(\min\left(D^2N, N^2D\right))$ operations [49].

The P-IPCAC's training phase requires at most $O(M_A M_B \min\left(D^2N, N^2D\right))$. This result underlines the linear dependency of the P-IPCAC's computational complexity with respect to the number of clusters, that seems to be unacceptable for high values of M_A and M_B; nevertheless, increasing the number of clusters reduces the number of points per cluster, thus allowing SVD to be more efficient.

The computational complexity of the P-IPCAC's classification phase is dominated by the evaluation of Equation (5.16), that is the product of the matrix $\boldsymbol{W}^T$, having $M_A M_B$ rows and D columns, times the D-dimensional point $\boldsymbol{p}$ to be classified, thus requiring $O(M_A M_B D)$ operations.

5.5 Experimental Results

To evaluate the P-IPCAC performances we executed three tests: two of them are applied on synthetic datasets (see Section 5.5.1), and one is applied on a real dataset taken from the UCI Machine Learning Repository [2] (see Section 5.5.2).

In order to show the effectiveness of our algorithm, we compared its results to those obtained by SVM; more precisely, we used the RBF kernel, and we tuned its hyper-parameters[5] (C, σ) by using the grid optimization with 2-fold cross-validation on the whole dataset. Moreover, we compared our results with those obtained by employing $\texttt{IPCAC}_{var}$[6], its kernel version K-IPCAC, and K-TIPCAC, appropriately tuned.

To evaluate and compare the tested classifiers, we adopted the Accuracy, Sensitivity, and Specificity measures described in Appendix A.

All the described experiments were executed by using a MATLAB implementation of all the tested algorithms[7]; moreover we used a Linux 2.6.27 operating system running on an Intel Core(2) Duo T9400 CPU, equipped with 4GB of RAM.

5.5.1 Experiments on Synthetic Data

To evaluate our classifier, we performed our tests on 500-dimensional synthetic data generated by two different sample generators. To obtain a robust estimation of the evaluation parameters (*Accuracy*, *Specificity*, and *Sensitivity*), and to estimate the Execution Time[8], we averaged the results obtained by running each test 10 times, and by using 10-fold cross-validation.

[5] The hyper-parameter C represents the cost-value, while σ is the RBF kernel-radius.

[6] We retain the 99% of the variance on the syntethic tests, and the 99.9% of the variance on the real data tests.

[7] For the SVM classifier we used a Matlab wrapper to the library "libSVM" [12].

[8] Execution Time = Training time + Classification time.

First Generator

The datasets produced by this generator were composed by 500-dimensional points drawn from 4 MGDs per class. Each MGD was an isotropic Gaussian with standard deviation e, where $e \in \{1, 2, 3\}$ for the first, the second, and the third experiment respectively. In these datasets the overlap between the classes increases with the parameter e, thus producing datasets representing classification problems with increasing difficulty.

The MGD means of the first class were set to:

- $(0, \ldots, 0)^T$: a vector with 500 zeros;
- $(1, \ldots, 1, 0, \ldots, 0)^T$: a vector with 250 ones and 250 zeros;
- $(0.5, \ldots, 0.5, -1, \ldots, -1)^T$: a vector with 250 values fixed to 0.5 and 250 values fixed to -1;
- $(0.5, \ldots, 0.5, 1, \ldots, 1)^T$: a vector with 250 values fixed to 0.5 and 250 ones.

Whilst, the means of the MGDs used for the second class were:

- $(1, \ldots, 1)^T$: a vector with 500 ones;
- $(1, \ldots, 1, -1, \ldots, -1)^T$: a vector with 250 ones and 250 values fixed to -1;
- $(0, \ldots, 0, 1, \ldots, 1)^T$: a vector with 250 zeros and 250 ones;
- $(0, \ldots, 0, -1, \ldots, -1)^T$: a vector with 250 zeros and 250 values fixed to -1.

The sample generator randomly drew 500 points from each MGD, thus obtaining two classes with 2000 (500-dimensional) points each.

As shown in Table 5.4, IPCAC$_{var}$ obtained the worst results; this is due to the fact that the considered classes are not linearly separable. On the other side, P-IPCAC demonstrated its effectiveness by achieving the highest accuracy. Moreover, the Execution Time of our algorithm was considerably lower in comparison with the kernel methods, confirming the efficiency of the proposed technique.

Dataset	*Classifier*	*Acc. (std)*	*Spec.*	*Sens.*	*Time*
1	IPCAC$_{var}$	51.50 (3.08)	51.76	51.24	1.6
	SVM	98.38 (0.64)	98.86	97.90	1443.8
	K-IPCAC	96.08 (1.16)	94.76	97.40	487.8
	K-TIPCAC	97.85 (0.96)	97.30	98.40	493.8
	P-IPCAC	***99.85 (0.17)***	***99.91***	***99.79***	***170.1***
2	IPCAC$_{var}$	50.28 (3.44)	50.00	50.56	1.6
	SVM	75.98 (1.92)	79.99	71.97	1513.4
	K-IPCAC	76.10 (1.91)	78.11	74.09	497.8
	K-TIPCAC	77.08 (1.96)	77.87	76.29	498.9
	P-IPCAC	***92.20 (1.11)***	***92.63***	***91.77***	***184.6***
3	IPCAC$_{var}$	49.95 (2.16)	50.63	49.27	1.6
	SVM	66.80 (2.73)	69.12	64.48	1513.9
	K-IPCAC	67.10 (1.87)	67.42	66.78	496.5
	K-TIPCAC	69.18 (1.71)	69.44	68.92	497.1
	P-IPCAC	***81.02 (2.17)***	***80.17***	***81.87***	***186.3***

Table 5.4: Experimental results obtained on the datasets generated with $e \in \{1, 2, 3\}$. The last column represents the average Execution Time (in seconds) for both the training and the classification phase.

Second Generator

A second generator was built to randomly produce data drawn from non-isotropic MGDs. To achieve this goal, it is necessary to randomly choose the parameters defining the Gaussian Distributions, that are the mean vectors and the covariance matrices.

For each MGD, the mean vector coordinates are randomly selected in the open interval $(0, 1)$ according to the uniform distribution, while the covariance matrix is obtained by randomly generating both the positive eigenvalues, and the orthonormal eigenvectors. More precisely, the eigenvalues were generated by employing the random generator for the uniform distribution, while the eigenvectors were drawn from the normal distribution and they were orthogonalized through the Graham-Schmidt orthogonalization process.

By employing this generator we produced three datasets, each composed by two classes containing 500-dimensional points:

Dataset1 was composed by a balanced number of points, drawn by a balanced number of MGDs. More precisely, each class contained 2000 points

taken from 4 MGDs, where 500 points were drawn from each Gaussian.

Dataset2 was composed by points drawn by an unbalanced number of MGDs. More precisely, class A was composed by 2000 points drawn from 4 MGDs, (500 points per MGD); whilst class B contained 1500 points drawn from 3 MGDs (500 points per MGD).

Dataset3 was composed by an unbalanced number of points, drawn by an unbalanced number of MGDs. More precisely, class A included 1400 points taken from 4 MGDs, where $\{200, 300, 400, 500\}$ points were drawn from each of them respectively; whilst class B contained 1200 points taken from 3 MGDs, where $\{300, 400, 500\}$ points were drawn from each of them respectively.

All the datasets were characterized by overlapping classes; moreover, they were built in order to be not linearly separable.

The results, reported in Table 5.5, are commented below:

- in all the experiments, P-IPCAC outperformed the other classifiers;
- P-IPCAC demonstrated to be robust to an unbalanced number of MGDs between the classes, as demonstrated by the experiments on Dataset2;
- P-IPCAC showed to be slightly sensitive to the unbalancing of the number of points between MGDs;
- regarding the Execution Time, P-IPCAC demonstrated its efficiency; indeed, only the IPCAC$_{var}$ algorithm was faster.

5.5.2 Experiments on a Real Dataset

The Hungarian heart disease dataset[9] is obtained from the UCI machine learning repository [2]. This dataset includes 106 feature vectors related to patients affected by heart disease, and 188 feature vectors related to patients without heart disease; moreover, 13 features describe each patient. In our experiments we used all the given feature vectors, including those containing

[9] http://archive.ics.uci.edu/ml/datasets/Heart+Disease

Dataset	*Classifier*	*Acc. (std)*	*Spec.*	*Sens.*	*Time*
1	IPCAC$_{var}$	91.22 (1.45)	91.29	91.15	1.6
	SVM	95.00 (1.39)	95.41	94.59	1818.8
	K-IPCAC	94.38 (1.16)	93.92	94.84	492.2
	K-TIPCAC	95.43 (1.21)	95.08	95.78	496.3
	P-IPCAC	***98.08 (0.63)***	***98.73***	***97.43***	***216.4***
2	IPCAC$_{var}$	93.94 (1.09)	93.27	94.61	1.5
	SVM	96.77 (0.40)	95.24	98.30	1049.1
	K-IPCAC	96.26 (0.80)	94.80	96.72	345.7
	K-TIPCAC	97.22 (0.60)	96.60	97.84	360.6
	P-IPCAC	***98.49 (0.74)***	***98.63***	***98.35***	***154.2***
3	IPCAC$_{var}$	94.23 (1.07)	94.10	94.36	1.5
	SVM	96.27 (1.47)	94.91	97.63	1422.3
	K-IPCAC	95.23 (1.11)	93.04	97.42	415.7
	K-TIPCAC	96.32 (1.01)	94.72	97.92	420.4
	P-IPCAC	***96.50 (0.93)***	***97.09***	***95.61***	***170.1***

Table 5.5: Experimental results obtained on the datasets produced by the second generator. The last column represents the average Execution Time (in seconds) for both the training and the classification phase.

missing attribute values; we completed the partially specified samples by using a fixed value for the missing attributes.

It is important to notice that this dataset does not fulfill the base assumption of P-IPCAC, becoming an important test to demonstrate the generalization capability of our method.

For the heart disease dataset, 50 runs of 10-fold cross-validation were performed, and the averaged *Accuracy*, *Specificity*, *Sensitivity* and Execution Time were computed and reported in Table 5.6.

As shown by the obtained results, P-IPCAC performs well also in this real case where the data are not drawn from MoGs. This fact, and the qualitative experiments reported in Figure 5.6, show the "weak" dependency of P-IPCAC from its assumptions. In our opinion, this is probably due to the fact that a large class of pdf can be approximated by means of Gaussian Mixtures, as reported in [32]. Furthermore, representing a multimodal pdf with a MoG guarantees a better approximation than that achieved by using a single MGD.

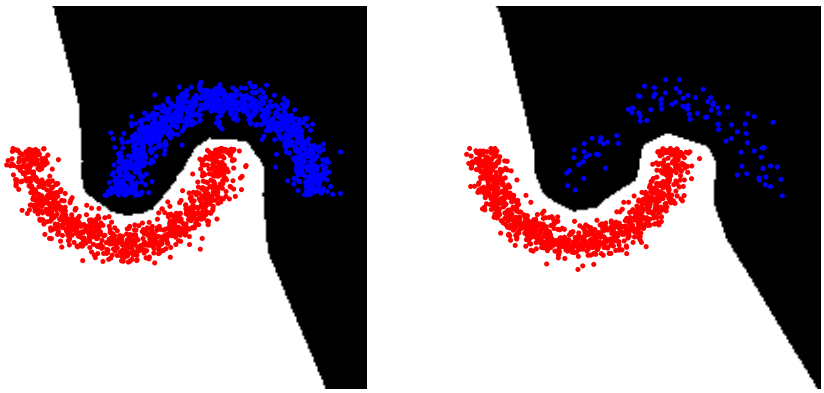
Finally, the Execution Time achieved by P-IPCAC was considerably lower than that of the kernel methods, confirming the results of the previous ex-

periments.

Classifier	*Accuracy (std)*	*Specificity*	*Sensitivity*	*Time*
IPCAC$_{var}$	78.96 (6.89)	69.58	88.34	3.3
SVM	80.19 (7.03)	75.72	84.66	129.4
K-IPCAC	81.20 (7.00)	77.28	85.12	162.7
K-TIPCAC	81.72 (7.00)	77.66	85.78	167.6
P-IPCAC	***81.92 (6.87)***	***77.42***	***86.42***	***19.0***

Table 5.6: Experimental results obtained on UCI heart diseases. The last column represents the average Execution Time (in milliseconds) for both the training and the classification phase.

Figure 5.6: *Qualitative experiments were executed on two synthetically generated datasets in* $\Re^2$. *The first experiment (left) contains balanced sets of training points (*1000 *per class); the second experiment (right) contains unbalanced sets of training points (*1000 *and* 100 *respectively). In both figures, white background represents points classified as belonging to the "red" class, whilst black background represents points belonging to the "blue" class. Notice that, intentionally, the generated classes do not fulfill the* MoGs *assumption.*



5.6 Conclusions

In this chapter we faced the problem of classes that are non linearly separable, presenting two algorithms (K-IPCAC and P-IPCAC) that generalize the IPCAC assumptions.

K-IPCAC is a kernel version of IPCAC that exploits the "kernel trick" to

perform non-linear classification. Furthermore, in this chapter we presented different improvements of K-IPCAC with the aim of reducing the overfitting problems that might affect kernel based techniques.

P-IPCAC starts from the base assumption that the feature vectors of each class are drawn from a Mixture of Gaussians (MoG). Moreover, the clusters representing the MoG components are automatically determined.

We evaluated the performances of the proposed algorithms by executing experiments both on synthetic and real datasets. These tests confirm that the proposed methods are promising.

Chapter 6

DDAG K-TIPCAC for Multiclass Classification

In this chapter we present an ensemble classifier combining several K-TIPCACs to perform multiclass classification. More precisely, we focus on the task of protein subcellular location prediction, which is one of the most difficult multiclass prediction problems in modern computational biology. Although many methods have been proposed in the literature to solve this problem, all the existing approaches are affected by some limitations so that the problem is still open. For this reason the method proposed in this chapter, called DDAG K-TIPCAC, combines several kernel classifiers through Decision Direct Acyclic Graph (DDAG). Experimental results clearly indicate that DDAG K-TIPCAC performs equally, if not better, than state-of-the-art ensemble methods aimed at multiclass classification of highly unbalanced data.

This chapter is organized as follows: in Section 6.1 the related works for protein subcellular localization are summarized; in Section 6.3 the experimental settings, our method, and the multiclass Fisher subspace estimated on the training vectors by means of a novel technique are described; in Section 6.4 the results obtained, compared with state-of-the-art ensemble methods, are reported, and further experiments are shown.

6.1 Problem Definition and Related Works

Since many different protein molecules are present in one or more subcellular locations, a better understanding of their distribution and function is advisable to understand the complex biological systems that regulate the biological life of each cell. To this aim, the first and fundamental problem to be solved is the subcellular protein localization. Since biochemical experiments aimed at this task are both costly and time-consuming, and new proteins are continuously discovered (increasing the gap between the newly found proteins and the knowledge about their subcellular location), an efficient and effective automatic method for protein subcellular location prediction is required.

This problem can be formulated as a multiclass classification problem as follows. The training dataset, $\boldsymbol{\mathcal{P}}_{Train}$, is composed of N protein vectors, $\boldsymbol{\mathcal{P}}_{Train} = \{\boldsymbol{p}_i\}_{i=1}^{N}$, where each protein sequence can be represented as a vector $\boldsymbol{p} = [R_s^j]$, R_s^j being the amino acid residue whose ordered position in the sequence is $s = 1, \cdots, S$ (S is the protein length, which differs in each protein), while the superscript $j = 1, \ldots, 20$ indicates which native amino acid is present in the s-th position of the sequence. The proteins in $\boldsymbol{\mathcal{P}}_{Train}$ are classified into M subsets $\boldsymbol{\mathcal{S}} = \bigcup_{i=1}^{M} \boldsymbol{\mathcal{S}}_i$, where each subset, $\boldsymbol{\mathcal{S}}_m$ ($m = 1, 2, \ldots, M$), is composed of proteins with the same subcellular component, and the cardinality of $\boldsymbol{\mathcal{S}}$ is $|\boldsymbol{\mathcal{S}}| = N = N_1 + N_2 + \cdots + N_M$. The classifier's aim is to learn the information provided by $\boldsymbol{\mathcal{P}}_{Train}$ to predict the subcellular location of a query protein $\boldsymbol{p}_q$.

In the past decade many authors have tried to handle this problem, and several classification methods have been proposed [21]. Nevertheless, the problem is still open due to several difficulties that make the task of protein subcellular location prediction very challenging. At first, the protein data are usually encoded with high dimensional vectors, so that the employed classifiers should be designed in order to minimize the computational complexity. Secondly, the number of subcellular locations that should be discriminated is at most 22 (that is $M \leq 22$), and some proteins, called multiplex proteins, might be present in more than one cellular component, or they might move from one location to another. Finally, the protein sub-

cellular distribution is highly unbalanced since some cellular components contain a significantly lower number of protein molecules.

To achieve satisfactory results in such (multiclass, high dimensional, and highly unbalanced) classification problem, a dataset of high cardinality is needed. Unfortunately, the training datasets have a limited number of proteins, due to the following reasons: some proteins must be discarded since they contain less than 50 amino acids, or they are annotated as 'fragments'; to avoid homology bias proteins with $\geq 25\%$ sequence identity to any other in the same subcellular organelle must be eliminated; proteins belonging to components with less than 20 proteins are generally excluded, because of lacking statistical significance; several proteins cannot be used as robust data for training a solid predictor since they have not been experimentally annotated yet. Finally, further deletions might be performed by some authors focusing on proteins with a unique subcellular location, or belonging to a specific organism. These difficulties motivate the large number of research works devoted to the task of protein location prediction; these methods can be grouped according to either the data representation, or the employed algorithm.

Representing a protein $\boldsymbol{p}$ with a vector that codes its entire amino acid sequence is unfeasible since this representation produces too long vectors of different dimensionality. A more compact representation is provided by the amino acid composition (`AAC`) descriptor [17], whose elements are the normalized occurrence frequencies of the 20 native amino acids. Since the `AAC` lacks the ability of representing the sequence order effects, several alternative non sequential descriptors have been proposed in the literature. More precisely, these descriptors, which represent both single and evolutionarily related groups of proteins, are:

- `PseAAC` encodes proteins by taking into account correlations between pairs of aminoacids at different sequence distance w.r.t a given chemico-physical property [18];
- the k-peptide encoding vector, which is the normalized occurrence of the k-letter pattern that appears in a window being shifted along the sequence, is another popular representation for single proteins [51];

- evolutionarily related groups of proteins can be encoded through the SeqEvo representation, based on the normalized occurrence of the changes in the protein sequence for each native amino acid (that is insertions, deletions, substitutions of amino acid residues) that are due to proteins evolution [16].

While the aforementioned protein representation schemes are all strictly based on the protein amino acid sequence, alternative encodings are possible by considering the availability of large amount of information contained in public databases like the Functional Domain (FunD) [19] and the Gene Ontology (GO) [23]. According to the content of FunD it is possible to code each protein in the form of a boolean vector indicating the presence/absence of any of the 7785 functional protein domains annotated in the database and a similar encoding scheme can be adopted by considering the annotations stored in the Cellular Component division of the Gene Ontology.

Regarding the employed predictors, they are: the Covariant Discriminant (CD) algorithm [18]; modified versions of the K-Nearest-Neighbor (KNN) technique [26, 51, 88], or its extension, called Optimized Evidence-Theoretic KNN (OET-KNN) [104, 20], Support Vector Machines (SVMs) [25, 58, 40], and the naive Bayes classifier [9]. All the aforementioned methods are depending on critical parameters, defining both the protein representation mode, the dataset dimensionality, and different settings of the learning algorithm. Recently, simple ensemble methods have been proposed: given an engine learning algorithm (e.g. OET-KNN or SVM), these techniques create different predictors by changing the values of their parameters, and produce the final classification result by a simple majority vote algorithm [20, 89, 16].

Although promising results have been obtained, the computational efficiency and the classification performance of all the above mentioned techniques are highly affected both by the high unbalancing of the training set, and by the low cardinality of some classes compared to the high data dimensionality.

To overcome such weaknesses, in this chapter we propose our ensemble method whose engine algorithm is the Kernel Truncated Isotropic Princi-

pal Component Analysis Classifier (K-TIPCAC, see Section 5.2.2), an evolution of the K-IPCAC and the O-IPCAC algorithms; it projects the points on the Fisher subspace estimated by a novel technique on the training data (see Section 4.3). The ensemble method combines the results computed by different K-TIPCAC predictors through a Decision Directed Acyclic Graph (DDAG) technique [78], as described in the following section.

6.2 DDAG K-TIPCAC

In this section we briefly describe the DDAG technique which allows to combine different methods to create a multiclass classifier (see Section 6.2.1). Secondly we describe how DDAG has been exploited to develop our technique, called DDAG K-TIPCAC (see Section 6.2.2).

6.2.1 Decision DAGs (DDAGs)

A Rooted Direct Acyclic Graph (DAG) is a graph whose edges have an orientation, no cycles, and only one root node. A Rooted Binary DAG has nodes which have either 0 or 2 arcs leaving them. A DDAG [78] is a method that combines the results of *one-against-one* classifiers to produce a multiclass classification. To this aim, considering a N-class problem, the DDAG is implemented using a rooted binary DAG with $K = N(N-1)/2$ internal nodes. Each node represents a classification model trained on two of the K classes, and it produces a boolean output value ($\{0, 1\}$). The nodes are arranged in a binary tree with the single root node at the top, two nodes in the second layer and so on until the final layer of leaves. Considering each classifiers as a boolean function, to perform a classification the DDAG proceeds as follows: it starts at the root node and it evaluates the boolean function; the node is then exited via the left edge, if the binary function is zero, or the right edge, if the binary function is one; the next nodes binary function is then evaluated; the membership class is the final leaf node reached through this process.

6.2.2 Decision DAG K-TIPCAC

In Section 4.3 and in Section 5.2.2, an efficient binary classifier (TIPCAC[1]) and its kernel version (K-TIPCAC) are described, that are based on the projection of the data on the one dimensional Fs estimated in a partially whitened kernel subspace.

The ensemble classifier proposed in this chapter is a C-class classifier that projects the data on a $C-1$ dimensional Fs estimated in a partially whitened subspace, and then applies DDAG to combine many binary K-TIPCACs (described in Chapter 5) to obtain the final prediction.

More precisely, the first step of this method evaluates the Fs of the overall C classes by generalizing the partial whitening approach recovering residuals, used by T-IPCAC and described in Chapter 4; accordingly to what observed in Section 4.3, this step reduces the training time complexity.
To this aim, after the partial whitening, the whitened class means $\{\boldsymbol{\mu}_c\}_{c=1}^{C}$ are computed as follows:

$$\boldsymbol{\mu}_c = \boldsymbol{W}_D\hat{\boldsymbol{\mu}}_c = q_d\boldsymbol{U}_d\boldsymbol{Q}_d^{-1}\boldsymbol{U}_d^T\hat{\boldsymbol{\mu}}_c + \hat{\boldsymbol{\mu}}_c - \boldsymbol{U}_d\boldsymbol{U}_d^T\hat{\boldsymbol{\mu}}_c.$$

At this stage the orthonormal basis, Π_{C-1}, composed of $C-1$ vectors spanning the Fs, is computed. More precisely, Π_{C-1} is obtained by orthonormalizing the $C-1$ linearly independent $\boldsymbol{\mu}_c$ vectors through the Gram-Schmidt procedure. The partially whitened training points $\boldsymbol{\mathcal{P}}_{\boldsymbol{W}_D}$ are then projected on the subspace Π_{C-1}, obtaining the set of points

$$\boldsymbol{\mathcal{P}}_{\Pi_{C-1}} = \left\{\boldsymbol{Fs}^T\boldsymbol{p}_i | \boldsymbol{p}_i \in \boldsymbol{\mathcal{P}}_{\boldsymbol{W}_D}\right\},$$

where $\boldsymbol{Fs}$ is the matrix whose columns span Fs.

Exploiting the points in $\boldsymbol{\mathcal{P}}_{\Pi_{C-1}}$, $C(C-1)/2$ K-TIPCAC binary classifiers are trained, each discriminating two classes in a *one-against-one* fashion (1-vs-1), and their results are combined by means of DDAG.

6.3 Experimental Setting

In this section we firstly remind the multiclass classification method employed to perform the base-line comparison (see Section 6.3.1); secondly,

[1]For simplicity the batch implementation of OIPCAC is called TIPCAC in the follows.

we describe in details the employed dataset (see Section 6.3.2); finally, we report the performance evaluation (see Section 6.3.3).

6.3.1 Methods

Multiclass Support Vector Machine: Since SVM is a binary classifier, a problem transformation is required before the application of this method to the considered multiclass prediction problem. The existing approaches to cast a multiclass classification problem to a series of binary classification problems can be roughly divided into two main classes: *one-against-all* and 1-vs-1. We applied the latter, and thus we trained a committee of 231 probabilistic SVMs [77]. The probabilities produced by each classifier were then reconciled to a multiclass prediction via pairwise coupling [47] and a simple max rule over all the class probability estimates was applied to make a final decision.

Ensemble of nested dichotomies (END): Nested dichotomies [34] is a standard statistical technique applied in polytomous classification problems where logistic regression is applied by fitting binary logistic regression models to the internal nodes composing a tree. In absence of domain knowledge it is difficult to decide, among all the possible trees of nested dichotomies, the one to be adopted. A possible solution [35] is to consider all the hierarchies of nested dichotomies equally likely, and to use an ensemble of these hierarchies for prediction. In our experiments we used the END implementation provided in WEKA and we tuned across nd (number of dichotomies) $\in \{5, 10, 20, 40\}$.

Random Forest (RF): Random Forest [8] has been applied as an effective tool for biomolecular and bioinformatics research. This method grows many classification trees. Instances whose class needs to be predicted are classified using the trees composing the forest. Each tree computes its own prediction, and the forest employs a plurality voting (over all the trees in the forest) to choose the final classification. We tuned the method using a grid search over nt (number of trees of the forest) $\in \{10, 20, 30, 40, 50\}$ and nf (number of features) $\in \{10, 100\}$.

Neighbors Voting (NV): The Tensor Voting Framework (TVF, [65]) is a manifold learning technique. More precisely, given a set of training vectors TVF is generally employed to estimate the normals to the manifolds underlyning them. This technique could be applied for classification by learning the manifold describing each class. Although effective, this method has a too high computational cost that makes its application unpractical. To overcome this limitation, in [39] the authors proposed a simplified version of TVF, called Neighbors Voting, that achieves a comparable normal estimation accuracy with a lower computational complexity.

Exploiting NV we have created a multiclass classifier based on the estimation of the 22 manifolds (one for each class) underlying the data projected into the $C-1$ dimensional subspace (estimated as explained in Section 6.2.2). Given a new point, this classifier assigns it to the class corresponding to the nearest estimated manifold. We extensively tuned the NV algorithm parameters, selecting the best values for: the neighborhood size (σ), the noise standard deviation (γ), and the number of iterations to be performed.

6.3.2 Dataset

We evaluated the proposed method on a publicly available dataset[2] involved in the training of the EukP-loc method described in [15].

This dataset contains 5618 different proteins, classified into 22 eukaryotic subcellular locations. Among the 5618 considered proteins, 5091 belong to one subcellular location, 495 to two locations, 28 to three locations, and 4 to four locations. None of the proteins has $\geq 25\%$ sequence identity to any other in the same subset. The collection of sequences was then evaluated to compute the Pseudo Amino Acid compositions (PseAAC) of each protein using the PseAAC web server [87]. For each protein we produced a 495-elements vector composed by 20 numbers describing the standard amino acid composition, 400 values representing the PseAAC based on the dipeptide representation of the protein and further 75 values representing three groups of 25 PseAACs values obtained by setting the λ parameter to 25 and computing the PseAACs based on three pairs of chemico-physical properties:

[2]The protein sequences were downloaded in fasta format from the web site http://www.csbio.sjtu.edu.cn/bioinf/euk-multi/Supp-A.pdf.

Dataset			
acrosome proteins	17	cell wall proteins	47
Golgi proteins	157	spindle pole body proteins	17
hydrogenosome proteins	13	synapse proteins	13
lysosome proteins	59	vacuole proteins	91
melanosome proteins	13	centriole proteins	45
microsome proteins	23	chloroplast proteins	497
mitochondrion proteins	488	cyanelle proteins	85
nucleus proteins	1077	cytoplasm proteins	741
peroxisome proteins	92	cytoskeleton proteins	46
plasma membrane proteins	647	endoplasmic reticulum proteins	275
extracell proteins	609	endosome proteins	39

Table 6.1: Protein subcellular localization prediction dataset (5091 proteins and 22 locations). This table reports the number of annotated proteins per location; labels are mutually exclusive, thus the problem is multiclass but not multilabel.

Hydrophobicity-Hydrophilicity, `pK1` (`alpha-COOH`)-`pK2` (`NH3`) and `Mass-pI`. In this preliminary investigation we focused on the location prediction of the 5091 proteins with a single experimentally annotated subcellular location. Some characteristics of this dataset are depicted in Table 6.1. It is worth noting that the problem is highly unbalanced, ranging the number of proteins associated to a subcellular location from 13 (hydrogenosome, melanosome and synapse) to 1077 (nucleus).

6.3.3 Performance Evaluation:

All the compared methods were evaluated according to a canonical 10 fold stratified cross-validation scheme. Given that the considered problem is a multiclass prediction problem affected by severe unbalance, accuracy is not suitable for performance evaluation. Performances were thus collected in form of F-score (that is the harmonic mean of Precision and Recall, as described in Appendix A). All the experiments, apart those involving the `DDAG K-TIPCAC`, and `NV`, which are implemented in MATLAB, were performed using the `WEKA` machine learning library [44].

Performance evaluation				
Method	Parameters	Precision	Recall	F-score
DDAG_K-TIPCAC	kernel=RBF, $\sigma = 8$, $var = 0.955$	0.383	0.408	**0.390**
Multiclass SVM	$C = 10.0$ $G = 0.01$	0.369	0.409	0.368
END	$nd = 40$	0.351	0.393	0.355
RF	$nt = 50$ $nf = 100$	0.349	0.391	0.340
NV	$\sigma = 2$ $\gamma = 0.2$ $iterations = 10$	0.348	0.348	0.347

Table 6.2: Estimated performances obtained by 10 fold stratified cross validation.

6.4 Results

The performances achieved by the evaluated approaches averaged across all the classes are reported in Table 6.2. The table shows, for each method, the best setting of its parameters, and the achived performance measures, that are the Precision, Recall, and F-measure. The F-scores obtained by the evaluated methods for each subcellular location averaged across the 10 stratified cross validation folds are reported in Table 6.3. In order to investigate if the differences between the collected per class performances are statistically significant we performed a Wilcoxon signed ranks sum (U) test [99]. Results are reported in Table 6.4 (direction of the comparison is row-vs-column).

Considering the performances averaged across all the classes achieved by the compared ensemble methods (see Table 6.2) the best performing approach is DDAG K-TIPCAC (weighted F-score 0.390) immediately followed by the 1-vs-1 ensemble of SVMs (weighted F-score 0.368). A closer look to this table highlights that, while all the evaluated approaches produced comparable Recall scores, on average this comes at the cost of a reduced precision, the only exceptions being represented by the DDAG K-TIPCAC ensemble and by the NV multiclass classifier. More precisely, while DDAG K-TIPCAC is the best performing classifier, NV obtains the significantly worst overall results.

We note that input space reduction is present in our approach and also in other types of ensembles evaluated in this experiment, as in the case of Random Forest. Nevertheless, the space reduction computed by RF might be affected by a more relevant information loss, since the input space dimensionality is reduced by means of a random selection of subsets of features of a priori defined size. We can hypothesize that the data transformation ap-

Per class performance evaluation (F-score)						
NV	END	MCSVM	RF	DDAG_K-TIPCAC	proteins	location
0.688	0.211	0.000	0.300	0.560	17	acrosome proteins
0.080	0.046	0.000	0.024	0.030	157	Golgi proteins
0.500	0.375	0.375	0.375	0.316	13	hydrogenosome proteins
0.241	0.000	0.000	0.033	0.213	59	lysosome proteins
0.818	0.632	0.000	0.556	0.522	13	melanosome proteins
0.129	0.000	0.000	0.000	0.114	23	microsome proteins
0.288	0.295	0.312	0.241	0.355	488	mitochondrion proteins
0.470	0.529	0.535	0.523	0.533	1077	nucleus proteins
0.166	0.000	0.000	0.000	0.047	92	peroxisome proteins
0.404	0.484	0.522	0.489	0.470	647	plasma membrane proteins
0.396	0.493	0.482	0.494	0.479	609	extracell proteins
0.309	0.175	0.218	0.157	0.267	47	cell wall proteins
0.000	0.000	0.000	0.000	0.306	17	spindle pole body proteins
0.519	0.700	0.700	0.700	0.383	13	synapse proteins
0.086	0.000	0.043	0.000	0.071	91	vacuole proteins
0.045	0.000	0.000	0.000	0.125	45	centriole proteins
0.440	0.424	0.504	0.459	0.518	497	chloroplast proteins
0.369	0.056	0.189	0.022	0.255	85	cyanelle proteins
0.274	0.247	0.235	0.211	0.290	741	cytoplasm proteins
0.070	0.000	0.000	0.000	0.059	46	cytoskeleton proteins
0.185	0.143	0.159	0.027	0.236	275	endoplasmic reticulum proteins
0.060	0.000	0.000	0.000	0.067	39	endosome proteins

Table 6.3: Per class performances obtained by 10 fold stratified cross validation.

Per class performance evaluation					
	END	MCSVM	RF	DDAG_K-TIPCAC	NV
END	–	0.6876	0.1317	0.9970	0.9918
MCSVM	0.3375	–	0.1813	0.9950	0.9780
RF	0.8826	0.8348	–	0.9874	0.9950
DDAG_K-TIPCAC	$2.689E^{-05}$	$3.073E^{-05}$	$4.449E^{-05}$	–	0.6869
NV	0.0091	0.0238	0.0056	0.3247	–

Table 6.4: Statistical comparison of per class performances through Wilcoxon test (alternative hypothesis: "greater", direction of comparison: rows versus columns).

plied by our approach is able to produce a more informative representation of the data than feature selection, thus leading to better performances also in highly unbalanced multiclass classification problems, as the one involved in our experiments.

This interpretation is supported by the collected per class performances

Performance evaluation				
Method	Parameters	Precision	Recall	F-score
DDAG_K-TIPCAC	Fs = "standard" kernel=RBF, $\sigma = 8$, $var = 0.955$	0.330	0.344	0.334

Table 6.5: Estimated performances obtained by 10 fold stratified cross validation and employing the projection on the multiclass Fs estimated with the "standard" methodology.

(see Table 6.3). As we can see, despite the multiclass SVM ensemble (MCSVM) ranks second in terms of overall F-score (after a weighted averaging of the per class F-scores), its performances are often worse that those obtained by DDAG K-TIPCAC. Moreover, it is important to highlight that all the methods misclassify at least one class, the only exception being represented by the DDAG K-TIPCAC approach. Note that NV obtains per class results comparable to DDAG K-TIPCAC; this is probably due to the fact that it employs the same input space reduction (on the Fisher subspace) employed in our method. This fact suggests that the method employed to estimate the Fisher subspace is promising, as further demonstrated by the experiments reported in the next subsections. The hypothesis that the performances, on a per class basis, of DDAG K-TIPCAC are better than those produced by most of the other evaluated methods is also supported by the Wilcoxon signed ranks sum test (see Table 6.4).

6.4.1 DDAG K-TIPCAC Employing the Standard Multiclass Estimation of Fs

In this section we want evaluate the effectiveness of our "truncated" approach to estimate the multiclass Fisher subspace. To this aim, we have performed the same experiment described in Section 6.3 by employing the points projected on the 21 dimensional Fisher subspace, as described in Section 3.2.1 spanned by the 21 eigenvectors corrisponding to the 21 largest eigenvalues of $\mathbf{\Sigma}_W^{-1}\mathbf{\Sigma}_{Bet}$.

In Table 6.5 the achieved overall performance is reported. It is possible to notice that the quality of the classification strongly decreases, obtaining results comparable with those achieved by RBF (see Table 6.2). In Table 6.6, where the per class F-measures are shown, we can note that some classes are completely misclassified. This demonstrates that the estimation of the mul-

Per class performance evaluation					
F-score	proteins	location	F-score	proteins	location
0.372	17	acrosome proteins	0.289	47	cell wall proteins
0.044	157	Golgi proteins	0.081	17	spindle pole body proteins
0.363	13	hydrogenosome proteins	0.333	13	synapse proteins
0.000	59	lysosome proteins	0.027	91	vacuole proteins
0.411	13	melanosome proteins	0.110	45	centriole proteins
0.000	23	microsome proteins	0.456	497	chloroplast proteins
0.290	488	mitochondrion proteins	0.110	85	cyanelle proteins
0.500	1077	nucleus proteins	0.226	741	cytoplasm proteins
0.053	92	peroxisome proteins	0.027	46	cytoskeleton proteins
0.474	647	plasma membrane proteins	0.186	275	endoplasmic reticulum proteins
0.334	609	extracell proteins	0.000	39	endosome proteins

Table 6.6: `DDAG_K-TIPCAC` per class F-measures obtained by 10 fold stratified cross validation employing the projection on the multiclass `Fs` estimated with the "standard" methodology.

Performance evaluation				
Method	Parameters	Precision	Recall	F-score
`DDAG_K-TIPCAC`	`Fs` = No kernel=`RBF`, $\sigma = 8$, $var = 0.955$	0.398	0.419	0.394

Table 6.7: Estimated performances obtained by 10 fold stratified cross validation and without employing the projection on the multiclass `Fs`.

ticlass Fisher subspace described in Section 6.2.2 is less affected by relevant information loss.

Concluding, the results confirm the quality of the proposed approach included the importance of the novel estimation of the multiclass Fisher subspace.

6.4.2 `DDAG K-TIPCAC` without Projection on Multiclass `Fs`

In this section we have performed another test using the same experimental setting described in Section 6.3. More precisely, we have eliminated the projection on the multiclass Fisher subspace, maintaining as engine classifier `K-TIPCAC` and then combining the binary classifiers to obtain the final prediction using `DDAG` methodology. This allows to evaluate the difference between this approach and the base ensemble method proposed in Section 6.2.2. The achieved overall performance is summarized in Table 6.7, while the per class results are reported in Table 6.8.

Even though the overall results obtained are slightly higher than those shown in Table 6.2 the computational cost of the technique employed in

Per class performance evaluation					
F-score	proteins	location	F-score	proteins	location
0.714	17	acrosome proteins	0.253	47	cell wall proteins
0.056	157	Golgi proteins	0.000	17	spindle pole body proteins
0.471	13	hydrogenosome proteins	0.538	13	synapse proteins
0.282	59	lysosome proteins	0.055	91	vacuole proteins
0.700	13	melanosome proteins	0.098	45	centriole proteins
0.000	23	microsome proteins	0.538	497	chloroplast proteins
0.344	488	mitochondrion proteins	0.167	85	cyanelle proteins
0.537	1077	nucleus proteins	0.290	741	cytoplasm proteins
0.019	92	peroxisome proteins	0.078	46	cytoskeleton proteins
0.489	647	plasma membrane proteins	0.247	275	endoplasmic reticulum proteins
0.477	609	extracell proteins	0.000	39	endosome proteins

Table 6.8: DDAG-K-TIPCAC per class F-measures obtained by 10 fold stratified cross validation without employing the projection on the multiclass Fs.

this subsection is too high. Anyhow, we would like to highlight that this performance confirms the quality of the K-TIPCAC engine algorithm.

Furthermore, considering the per class performance (reported in Table 6.8) we notice that 3 classes are completely misclassified, while the method proposed in Section 6.2.2 succeeds to identify all the 22 classes.

6.5 Conclusion

In this chapter we proposed an ensemble method whose engine algorithm is K-TIPCAC. The K-TIPCAC method deals with the points projected on a multiclass Fisher Subspace estimated extending the approach proposed in Chapter 4. The final multiclass classification is performed by combining the kernel classifiers through Direct Decision Acyclic Graph (DDAG).

This methodology was applied to one of the most difficult multiclass prediction problems in modern computational biology: the protein subcellular location prediction. The performed experimental tests shown the effectiveness of the K-TIPCAC algorithm and the quality of the multiclass Fisher Subspace estimation by means of the proposed approach.

It is worth noting that the ability of the proposed approach to effectively control the precision-recall trade-off also in the prediction of small classes is of paramount importance in real applications, when we need to reduce the costs associated with the biological validation of new protein locations

discovered through in silico methods.

Furthermore, considering also the experiments in Sections (6.4.1,6.4.2) we can affirm that the method proposed in this chapter is an efficient and effective technique. This is due to the fact that it outperforms state-of-the-art ensemble methods, and it reduces step the time cost employing a dimensionality reduction that is less affected by loss of discriminative information. Moreover, only this approach guarantees the identification of all the classes that describe the protein localization.

Chapter 7

Summary, Conclusions and Future Works

In this work we propose novel classifiers based on a new approach to estimate the Fisher Subspace. We face different classification tasks with the aim to stress our methods, to demonstrate their quality and to show that these classifiers solve the following problems:

1. Manage unbalanced classes;
2. Solve the small sample size problem;
3. Deal with high dimensional data;
4. Manage classification tasks where the space dimensionality is approximately equal to the cardinality of the training set;
5. Relax the linear separability constraint reducing the overfitting problems.

In Chapter 2 well-known learning techniques and some mathematical and geometrical tools used in the thesis are defined.

In Chapter 3 the base algorithm (`IPCAC`) is described in detail, reporting the results achieved on different datasets, furthermore, a Model Merging technique (`MM-IPCAC`) to merge different `IPCAC` trained models is proposed, and an improvement of `IPCAC` that performs an implicit dimensionality reduction to guarantee better performance is described.

The promising results achieved by the proposed classification algorithms (and their compact trained models), and their comparison with those obtained by well-known classification methods suggest that these techniques can be successfully applied as a basis for more complex learning algorithms.

In Chapter 4 we described the Online `IPCAC` linear binary classifier (`O-IPCAC`), an efficient and effective method that deals with dynamically supplied data in high dimensional spaces. This technique performs a "partial" whitening step recovering the residuals. More precisely, the data whitening has been replaced by a process that whitens the data in a linear subspace $\pi_d = \mathbf{Span}\langle \boldsymbol{v}_1, \ldots, \boldsymbol{v}_d \rangle, d \ll D$, while maintaining unaltered the information related to the orthogonal subspace $(\pi_d)^{\perp} = \mathbf{Span}\langle \boldsymbol{v}_{d+1}, \ldots, \boldsymbol{v}_D \rangle$.

`O-IPCAC` has been developed to deal with: high dimensional data, classification problems where the cardinality of the point set is high or the data are dynamically supplied, and highly unbalanced training sets whose cardinality is lower than the space dimensionality.

We evaluated the performance of this algorithm by executing experiments on `EEG` data, and on different real and syntethic datasets. It is important to underline that, in `EEG` classification, instead of focusing on complex features extraction/selection techniques, we propose a classifier that is able to deal with the employed raw data achieving good results.

In Chapter 5 we faced the problem of classes that are non linearly separable, presenting two algorithms (`KIPCAC` and `PIPCAC`) that generalize the `IPCAC` assumptions.

`K-IPCAC` is a kernel version of `IPCAC` that exploits the "kernel trick" to perform non-linear classification. Furthermore, in this chapter we presented different improvements of `KIPCAC` with the aim of reducing the overfitting problems that might affect the kernel based techniques.

`PIPCAC` starts from the base assumption that the feature vectors of each class are drawn from a Mixture of Gaussians (`MoG`). Moreover, the clusters representing the `MoG` components are automatically determined.

We evaluated the performances of the proposed algorithms by executing experiments both on synthetic and real datasets. These tests confirm the excellence of the proposed methods.

In Chapter 6 we proposed an ensemble method whose engine algorithm is an extension of KIPCAC (K-TIPCAC). The K-TIPCAC method deals with points projected on a multiclass Fisher Subspace estimated, from the training set, extending the approach proposed in Chapter 4. Multiclass classification is performed by combining the kernel classifiers through Direct Decision Acyclic Graph (DDAG).

This methodology was applied to one of the most difficult multiclass prediction problems in modern computational biology: the protein subcellular location prediction. The achieved results show the effectiveness of both K-TIPCAC technique and of the multiclass Fisher Subspace estimation method.

It is worth noting that the ability of the proposed approach to effectively control the precision-recall trade-off also in the prediction of small classes is of paramount importance in real applications, when we need to reduce the costs associated with the biological validation of new protein locations discovered through in silico methods.

Considering that most of the compared methods failed completely to predict the membership of proteins to particularly difficult subcellular locations, we conclude that DDAG K-TIPCAC is a promising line of research in this application domain.

In conclusion, the proposed classifiers, which are also described in [81, 82, 84, 83] obtain results that are usually better that those achieved by well-known classifiers. These results are of particular importance because they demonstrate that our methods are valuable tools for real data classification, and they offer the possibility to be employed for a wide amount of applications.

In future works we want to further investigate the PIPCAC algorithm; more precisely, we will focus on the preprocessing step, to improve the per class cluster estimation. Moreover, we plan to reduce its classification time complexity in two ways:

1. Choosing/designing a better clustering algorithm that reduces the number of discovered MoG components without loosing generality;

2. Pruning the first hidden layer by removing the useless classifiers, and merging neurons with similar behavior through the `MM-IPCAC` algorithm.

Furthermore, we want develop a time efficient technique to evaluate (a-priori) the best value for the number of d eigen values employed for the "partial" whitening in `O-IPCAC` (see Section 4.3), and we plan to prove the mistake bound of the on-line learning algorithm.

Another improvement that we would make is the Online version of the `K-TIPCAC` technique, proposing an adaptive kernel-version that copes with the possibility that the probability distribution underlying the data changes with time.

Regarding the real applications, we want to test our methods to datasets characterized by a very large ratio between the space dimensionality and the number of training points, such as Microarray data, and we plan to extend the `DDAG K-TIPCAC` approach for the multiplex classification, to provide a deeper characterization of its performances in further investigations on the protein localization problem.

Appendix A

Performance Evaluation Measures

Considering a *confusion matrix* for a dichotomic problem, the test results can be subdivided in 4 categories as shown in table A.1. The columns refer to the true (expected) values and the rows to the predicted values. P stands for *Positive* examples and N stands for *Negative* examples. True Positives (TP) are Positive examples correctly classified as Positive; True Negatives (TN) are Negative examples correctly classified as Negative; False Positives (FP) are Negative examples incorrectly classified as Positive, and False Negatives (FN) are Positives incorrectly classified as Negative. Using these distinct notions of correct and incorrect classification, we can define different quantities to evaluate the performance of a classifier.

Table A.1: Confusion matrix for a dichotomic problem.

		Expected	
		P	N
Predicted	P	TP	FP
	N	FN	TN

The *Sensitivity* expresses the ratio between the correctly predicted Positive examples and the total number of the Positive examples:

$$Sensitivity = \frac{TP}{TP + FN} \tag{A.1}$$

In the literature this quantity is also called *Recall*.
The *Specificity*, also called *True Negative Rate*, expresses the ratio between

the correctly predicted Negative examples and the total number of the Negative examples:

$$Specificity = \frac{TN}{TN + FP} \tag{A.2}$$

The *Precision* expresses the ratio between the correctly predicted Positive examples and the total number of examples predicted as Positive:

$$Precision = \frac{TP}{TP + FP} \tag{A.3}$$

The complementary of the Specificity (*1 - Specificity*) is the ratio between the examples incorrectly predicted as Positive and the total number of Negative examples, i.e. it expresses the fraction of the incorrectly classified Negative examples with respect to the total number of Negative examples:

$$1 - Specificity = 1 - \frac{TN}{TN + FP} = \frac{FP}{TN + FP} \tag{A.4}$$

Using the above notation, the *Accuracy* is the ratio between the number of correctly classified examples and the total number of examples:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{A.5}$$

To obtain a single evaluation measure that accounts of both *Recall* and *Precision* values, the weighted harmonic mean of precision and recall, that is called *F-measure* or balanced *F-score*, can be computed as follows:

$$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{A.6}$$

A.0.1 Basic concepts of ROC curve

The *Receiver Operating Characteristics* (ROC) curve has been introduced by the signal processing community in order to evaluate the capability of a human operator to distinguish informative radar signals from noise [31]. At the present, it is mostly used in the medical decision making community for assessing the usefulness of a diagnostic test.
In order to express in a synthetic way the performance of a classifier system, the Receiver Operating Characteristic (ROC) analysis offers a suitable tool to jointly evaluate sensitivity and specificity: it can be understood as a plot of the probability of classifying correctly the Positive examples against the rate of incorrectly classifying True negative examples. In this sense, one can interpret this curve as a comparison of the classifier across the entire range of class distributions and error costs. In ROC analysis the performance of a classifier is defined through pairs of Sensitivity and 1-Specificity values.

Hence, in this two-dimensional ROC space the performance of a certain classifier is defined by a point, i.e. by its 1-Specificity (X-axis) and Sensitivity (y-axis). In the case of classifiers obtained by thresholding, such as IPCAC or SVM, the ROC curve can be computed by varying the decision threshold of the classifier, which describes the trade-off between Specificity and Sensitivity. Using ROC curves the performance of different learning systems can be compared: the best point in the ROC plane is $(0, 1)$, i.e. $1 - Specificity = 0$ and $Sensitivity = 1$; the worst point is the opposite $(1, 0)$; ROC curves lying near the diagonal correspond to random guessing classifiers, and in general learning systems with ROC curves lying on the the top and leftmost portion of the ROC plane are the better ones. Figure (A.1) depicts an example of the ROC curve of a given classifier.
The most frequently used performance measure extracted from the ROC curve is the value of the *Area Under the Curve*, commonly denoted as AUC. When AUC is equal to 1, the classifier achieves perfect accuracy if the threshold is correctly chosen, and a classifier that predicts the class at random has an associated AUC of 0.5.

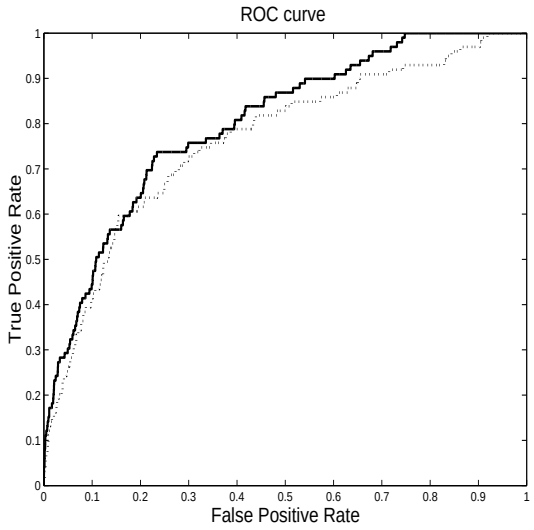


Figure A.1: *Example of ROC obtained with KIPCAC. The solid curve correspond to a gaussian kernel and the dotted one is the ROC curve of a polynomial kernel.*

Bibliography

[1] Project SpamAssassin Apache. Public spamassassin corpus, 2002-2005. Download Page at `http://spamassassin.apache.org/publiccorpus/`.

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] Peter N. Belhumeur, Joo P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, August 1997.

[4] Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Enrico Blanzieri and Anton Bryl. A survey of learning-based techniques of email spam filtering. DIT-06-056, January 2008. University of Trento.

[6] R. B. Bradford. An empirical study of required dimensionality for large-scale latent semantic indexing applications. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, 2008.

[7] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and Its Applications*, 415(1):20–30, 2006.

[8] L. Breiman. Random forests. *Machine Learning 45(1)*, 2001.

[9] S. Briesemeister, J. Rahnenfuhrer, and J. Kohlbacher. Going from where to why - interpretable prediction of protein subcellular localization. *Bioinformatics*, 2010.

[10] S. Charles Brubaker and Santosh Vempala. Isotropic PCA and affine-invariant clustering. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:551–560, 2008.

[11] N. Cesa-Bianchi, A. Conconi, and C. Gentile. A second-order perceptron algorithm. *SIAM J. Comput.*, 2005.

[12] Chih Chang and Chih Lin. LIBSVM: a library for support vector machines, 2001.

[13] L. Chen, H. Liao, M. Ko, J. Lin, and G. Yu. A new LDA-based face recognition system which can solve the small sample size problem. *Pattern Recognition*, 30:1713–1726, 2000.

[14] K.H. Chiappa. Evoked potentials in clinical medicine. In *Lippincott-Raven*, 1997.

[15] K. Chou and H. Shen. Cell-ploc: a package of web servers for predicting subcellular localization of proteins in various organisms. *Nature protocol*, 2008.

[16] K. Chou and H. Shen. A new method for predicting the subcellular localization of eukariotic proteins with both single and multiple sites: Euk-mPLoc. *Plos One*, 2010.

[17] K.C. Chou. A novel approach to predicting protein structural classes in a (20-1)-D amino acid composition space. *Proteins: Structure, Function, and Genetics*, 1995.

[18] K.C. Chou. Prediction of protein cellular attributes using pseudo amino acid composition. *Proteins: Structure, Function, and Genetics*, 2001.

[19] K.C. Chou and K.C. Cai. Using functional domain composition and support vector machines for prediction of protein subcellular location. *Journal of Biological Chemistry*, 2002.

[20] K.C. Chou and H.B. Shen. Predicting eukaryotic protein subcellular locations by fusing optimized evidence-theoretic K-nearest neighbor classifiers. *Journal of Proteome Research 5*, 2006.

[21] K.C. Chou and H.B. Shen. Recent progress in protein subcellular location prediction. *Analitical Biochemistry*, 2007.

[22] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, June 2009.

[23] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genet.*, 25:25–29, 2000.

[24] V. Gordon Cormack and Thomas R. Lynam. Spam corpus creation for TREC. In *CEAS*, 2005.

[25] C. Cortes and V. Vapnik. Support vector networks. *Machine learning*, 1995.

[26] T.M Cover and P.E. Hart. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory*, 1967.

[27] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *JMLR*, 7, 2006.

[28] Andrew Moore Dan Pelleg. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, 2000. Morgan Kaufmann.

[29] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley-Interscience Publication, 2000.

[30] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Recognition, 2nd edition.* Wiley-Interscience, New York, 2001.

[31] J. Egan. Signal decision theory and ROC analysis. *Academic Press*, 1975.

[32] Mario A.T. Figueiredo. On gaussian radial basis function approximations: Interpretation, extensions, and learning strategies. *Pattern Recognition, International Conference on*, 2:2618, 2000.

[33] William H. Press Saul A. Teukolsky William T. Vetterling Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[34] J. Fox. Applied regression analysis, linear models, and related methods. *Sage*, 1997.

[35] E. Frank and S. Kramer. Ensembles of nested dichotomies for multi-class problems. *Proceedings of the 21st ICML*, 2004.

[36] J.H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84:165–175, 1989.

[37] K. Fukunaga and R.R. Hayes. Effects of sample size in classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):873–885, 1989.

[38] Keinosuke Fukunaga. *Introduction to statistical pattern recognition (2nd ed.).* Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[39] P. Campadelli G. Lombardi, E. Casiraghi. The neighbors voting algorithm. *ECAI 2008 - Workshop on Supervised and Unsupervised Ensemble Methods and their Applications*, May 2008.

[40] A. Garg, M. Bhasin, and G.P. Raghava. Support vector machine-based method for subcellular localization of human proteins using amino acid compositions, their order, and similarity search. *Journal of Biological Chemistry*, 2005.

[41] C. Gentile. A new approximate maximal margin classification algorithm. *JMLR*, 2002.

[42] J. Grim and J. Hora. Iterative principles of recognition in probabilistic neural networks. *Neural Networks*, 21(6):838–846, 2008.

[43] R. Haeb-Umbach, D. Geller, and H. Ney. Improvements in connected digit recognition using linear discriminant analysis and mixture densities. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 2:239–242, 1993.

[44] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 2009.

[45] Peter Hall, David Marshall, and Ralph Martin. Merging and splitting eigenspace models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:2000, 1998.

[46] P. C. Hansen. The truncated SVD as a method for regularization. Technical report, Stanford University, Stanford, CA, USA, 1986.

[47] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *Proceedings of Adv. in Neural Information Processing Systems*, 1998.

[48] K. Hild, M. Kurimo, and V. Calhoun. The sixth annual mlsp competition. In *MLSP '10*, Sept. 2010.

[49] Michael Holmes, Alexander Gray, and Charles Isbell. Fast SVD for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*, 2007.

[50] Z.Q. Hong and J.Y. Yang. Optimal discriminant plane for a small number of samples and design method of classifier on the plane. *Pattern Recognition*, 24:317–324, 1998.

[51] Y. Huang and Y. Li. Prediction of protein subcellular locations using fuzzy K-NN method. *Bioinformatics*, 2004.

[52] Diaz. J., F. Diaz, C. Hernandez, E.M. Rodriguez, C. Diaz, and L. Serra. Application of linear discriminant analysis to the biochemical and hematological differentiation of opiate addicts from healthy subjects: a case-control study. *Eur. J. Clin. Nutr*, 58:449–455, 2004.

[53] Anil K. Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 4–37, 2000.

[54] Hua Yu Jie, Hua Yu, and Jie Yang. A direct LDA algorithm for high-dimensionaldata – with application to face recognition. *Pattern Recognition*, 34:2067–2070, 2001.

[55] I. M Johnstone and A. Y. Lu. Sparse principal components analysis. *Journal of the American Statistical Association*, 2004.

[56] T. Kim, S. Wong, B. Stenger, J. Kittler, and R. Cipolla. Incremental linear discriminant analysis using sufficient spanning set approximations. In *CVPR*. IEEE Computer Society, 2007.

[57] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, pages 259–284, 1998.

[58] Z. Lei and Y. Dai. An SVM-based system for predicting protein subnuclear localizations. *BMC Bioinformatics*, 2005.

[59] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Using discriminant analysis for multi-class classification: an experimental investigation. *Knowl. Inf. Syst.*, 10(4):453–472, 2006.

[60] K. Liu, Y. Cheng, and J. Yang. Algebraic feature extraction for image recognition based on an optimal discriminant criterion. *Pattern Recognition*, 26:903–911, 1993.

[61] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi. A review of classification algorithms for eeg-based brain-computer interfaces. *Journal of neural engineering*, 4(2), June 2007.

[62] Julie B. Lovins. Development of a stemming algorithm. *Massachusetts Inst of Tech Cambridge Electronic Systems Lab*, June 1968.

[63] Juwei Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. Regularized discriminant analysis for the small sample size problem in face recognition. *Pattern Recognition Lett.*, 24(16):3079–3087, 2003.

[64] Juwei Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. Regularization studies of linear discriminant analysis in small sample size scenarios with application to face recognition. *Pattern Recogn. Lett.*, 26(2):181–191, 2005.

[65] G. Medioni et al. Tensor voting: Theory and applications. *Proceedings of the French conference on Pattern Recognition. and Artificial Intelligence (RFIA)*, 2000.

[66] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.R. Muller. Fisher discriminant analysis with kernels. *Neural Networks for Signal Processing*, 1999.

[67] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.R. Muller. Constructing descriptive and discriminative onlinear features: Rayleigh coefficients in kernel feature spaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25, 2003.

[68] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[69] Javier M. Moguerza and Alberto Muñoz. Support vector machines with applications, Dec 2006.

[70] F. Orabona. *DOGMA: a MATLAB toolbox for Online Learning*, 2009. Software available at `http://dogma.sourceforge.net`.

[71] F. Orabona, C. Castellini, B. Caputo, J. Luo, and G. Sandini. Indoor place recognition using online independent support vector machines. In *BMVC '07*, pages 1090–1099, 2007.

[72] F. Orabona, J. Keshet, and B. Caputo. The projectron: a bounded kernel-based perceptron. *Int. Conf. on Machine Learning*, 2008.

[73] H.C. Palm. A new method for generating statistical classifiers assuming linear mixtures of gaussian densities. *Proceedings of the 12th IAPR conference: Computer Vision and Image Processing*, 2:483–486, 1994.

[74] D. Paul. Asymptotics of sample eigenstructure for a large dimensional spiked covariance model. *Statistica Sinica*, 2007.

[75] D. Paul. Asymptotics of sample eigenstructure for a large dimensional spiked covariance model. *Statistica Sinica*, 2007.

[76] T.W. Picton. The p300 wave of the human event related potential. In *Journal of Clinical Neurophysiology*, 1992.

[77] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. in Large Margin Classifiers, MIT press*, 1999.

[78] John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*, pages 547–553. MIT Press, 2000.

[79] J.J. Prieto, A. Talevi, and L.E. Bruno-Blanch. Application of linear discriminant analysis in the virtual screening of antichagasic drugs through trypanothione reductase inhibition. *Mol Divers*, 10(3):361–75, 2006.

[80] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988)).

[81] A. Rozza, G. Lombardi, and E. Casiraghi. Novel IPCA-based classifiers and their application to spam filtering. In *Proceedings of the 9th International Conference on Intelligent System Design and Applications (ISDA09)*. IEEE Computer Society, 2009.

[82] A. Rozza, G. Lombardi, and E. Casiraghi. PIPCAC: A novel binary classifier assuming mixtures of gaussian functions. In *AIA '10*. ACTA press, 2010.

[83] A. Rozza, G. Lombardi, M. Re, E. Casiraghi, and G. Valentini. DDAG K-TIPCAC: an ensemble method for protein subcellular localization. In *Proceedings of ECML PKDD - SUEMA 2010 workshop*. LNCS - Springer, 2010.

[84] A. Rozza, G. Lombardi, M. Rosa, and E. Casiraghi. O-IPCAC and its application to eeg classification. In *Proceedings of Workshop on Applications of Pattern Analysis (WAPA10)*. JMLR, 2010.

[85] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5):1299–1319, 1998.

[86] John Shawe-Taylor and Nello Cristianini. *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[87] H. Shen and K. Chou. PseAAC: a flexible web server for generating various kinds of protein pseudo amino acid composition. *Analytical Biochemistry*, 2008.

[88] H.B. Shen and K.C. Chou. Virus-PLoc: a fusion classifier for predicting the subcellular localization of viral proteins within host and virus-infected cells. *Biopolymers 85*, 2006.

[89] H.B. Shen and K.C. Chou. Hum-mPLoc: an ensemble classifier for large-scale human protein subcellular location prediction by incorporating samples with multiple sites. *Biochemical and biophysical research communications*, 2007.

[90] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT press, Massachusetts, 1998.

[91] Daniel L. Swets and John (Juyang) Weng. Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:831–836, 1996.

[92] Q. Tian, Z. Barbero, M.and Gu, and S. Lee. Image classification by the foley-sammon transform. *Optical Engineering*, 25:834–840, 1986.

[93] G. Valentini and T.G. Dietterich. Bias–variance analysis of support vector machines for the development of SVM-based ensemble methods. *Journal of Machine Learning Research*, 5:725–775, 2004.

[94] V. Vapnik. *Statistical Learning Theory*. Wiley Interscience, 1998.

[95] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[96] G. Voronoi. Nouvelles applications des parametres continus a la theorie des formes quadratiques. *Journal fur die Reine und Angewandte Mathematik*, 1907.

[97] G. Wahba. *Spline models for observational data*. SIAM, Philadelphia, USA, 1990.

[98] Q. Wei, Y. Wang, X. Gao, and S. Gao. Amplitude and phase coupling measures for feature extraction in an EEG-based brain computer interface. *Journal of Neural Engineering*, 2007.

[99] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[100] Jigang Xie and Zhengding Qiu. The effect of imbalanced data sets on LDA: A theoretical and empirical analysis. *Pattern Recogn.*, 40(2):557–562, 2007.

[101] Jing-Hao Xue and D. Michael Titterington. Do unbalanced data have a negative effect on LDA? *Pattern Recogn.*, 41(5), 2008.

[102] Jian Yang, Zhong Jin, Jing yu Yang, David Zhang, and Alejandro F. Frangi. Essence of kernel fisher discriminant: KPCA plus LDA. *Pattern Recognition*, 37(10):2097 – 2100, 2004.

[103] W. Zhao, R. Chellappa, and J. Phillips. Subspace linear discriminant analysis for face recognition. *Technical Report CS-TR4009, University of Maryland*, 1999.

[104] L.M. Zouhal and T. Denoeux. An evidence theoretic K-NN rule with parameter optimization. *IEEE Transactions on System, Man, and Cybernetics*, 1998.