A Tutorial on Distance Metric Learning: Mathematical Foundations, Algorithms and Software

Juan Luis Suárez jlsuarezdiaz@ugr.es
Department of Computer Sciences and Artificial Intelligence
University of Granada
C/ Periodista Daniel Saucedo Aranda, s/n, 18014 Granada, Spain

Salvador García salvagl@decsai.ugr.es
Department of Computer Sciences and Artificial Intelligence
University of Granada
C/ Periodista Daniel Saucedo Aranda, s/n, 18014 Granada, Spain

Francisco Herrera herrera@decsai.ugr.es
Department of Computer Sciences and Artificial Intelligence
University of Granada
C/ Periodista Daniel Saucedo Aranda, s/n, 18014 Granada, Spain

Abstract

This paper describes the discipline of distance metric learning, a branch of machine learning that aims to learn distances from the data. Distance metric learning can be useful to improve similarity learning algorithms, and also has applications in dimensionality reduction. We describe the distance metric learning problem and analyze its main mathematical foundations. We discuss some of the most popular distance metric learning techniques used in classification, showing their goals and the required information to understand and use them. Furthermore, we present a Python package that collects a set of 17 distance metric learning techniques explained in this paper, with some experiments to evaluate the performance of the different algorithms. Finally, we discuss several possibilities of future work in this topic.

Keywords: Distance Metric Learning, Classification, Mahalanobis Distance, Dimensionality, Similarity

1. Introduction

The use of distances in machine learning has been present since its inception. Distances provide a similarity measure between the data, so that close data will be considered similar, while remote data will be considered dissimilar. One of the most popular examples of this similarity learning is the well-known nearest neighbors rule for classification, where a new sample is labeled with the majority class within its nearest neighbors in the training set. This classifier was presented by Cover and Hart (1967), even though this idea had already been mentioned in earlier publications (see Sebestyen, 1962; Nilsson, 1965).

Algorithms in the style of the nearest neighbors classifier are among the main motivators of distance metric learning. These kind of algorithms have usually used a standard distance, like the euclidean distance, to measure the data similarity. However, a standard distance may ignore some important properties available in our dataset, so that the learning results could be non optimal. The search for a distance that brings similar data as close as possible, while moving non similar data away, can significantly increase the quality of these algorithms.

Distance metric learning has applications beyond the improvement of such algorithms. We will see, for example, that learning distances is closely related to learning linear mappings, which in turn is closely related with dimensionality reduction (see also Cunningham and Ghahramani, 2015).

Although techniques such as principal component analysis or linear discriminant analysis, which are considered distance metric learning techniques, have been popular in the statistical field since the middle of the 20th century, it is not until the beginning of the 21st century that distance metric learning is properly spoken of, and perhaps the algorithm from Xing et al. (2003) is responsible for drawing attention to this concept for the first time.

During the first decade of the 21st century some of the most well-known distance metric learning algorithms were developed, and are still popular today. The most relevant of these algorithms will be studied throughout this tutorial. Over recent years distance metric learning remains active, both in the search for new proposals for innovative distance metric learning algorithms, and in the refinement of techniques already employed over the past decade. Some of these techniques will also be shown.

In this paper we make a theoretical study of supervised distance metric learning, in which we show the mathematical foundations of distance metric learning and its algorithms. Furthermore, we analyze several distance metric learning algorithms for classification, from the problems and the objective functions they try to optimize, to the methods that lead to the solution of these problems.

Regarding the theoretical background of distance metric learning, we have studied three mathematical fields closely related with this topic. The first one is convex analysis (Rockafellar, 2015; Boyd and Vandenberghe, 2004). Convex analysis is present in many distance metric learning algorithms, since they try to optimize convex functions over convex sets. Some interesting properties about convex sets, as well as how to deal with constrained convex problems, will be shown in this study. We will also see how the use of matrices is a fundamental part of modeling our problem. Matrix analysis (Horn and Johnson, 1990) will therefore be the second field. The third field is information theory (Cover and Thomas, 2006), which is also used in some of the algorithms we will show. In addition, the theoretical approach on machine learning that we will follow is the one provided by Shalev-Shwartz and Ben-David (2014).

As explained before, our work focuses on supervised distance metric learning techniques. A large amount of algorithms have been proposed over the years. These algorithms were developed with different purposes and based on different ideas, so that we can classify them in different groups. In this way, we can find algorithms whose main goal is dimensionality reduction (Fisher, 1936; Wang and Zhang, 2007), algorithms specifically oriented to improve distance based classifiers, such as the nearest neighbors classifier (Weinberger and Saul, 2009; Goldberger et al., 2005), or the nearest centroid classification (Mensink et al., 2012), and a few techniques are also based on information theory (Davis et al., 2007; Nguyen et al., 2017; Globerson and Roweis, 2006). Some of these algorithms also allow kernel versions (Torresani and Lee, 2007; Mika et al., 1999; Wang and Zhang, 2007; Nguyen et al., 2017), that allow for the extension of distance metric learning to highly dimensional spaces.

We also present a Python package, called `pyDML`[1], that collects all the algorithms shown in this tutorial. This package, developed in the Python language, integrates the algorithms analyzed throughout the work, trying to provide extensive software on this subject, which is also compatible with the machine learning tools provided by Scikit-Learn (Pedregosa et al., 2011).

To complete this study, we carry out several experiments involving all the developed algorithms in pyDML executed over 34 datasets. For that, we define different settings to explore their performance and capabilities when considering maximum dimension, centroid-based methods, different kernels and dimensionality reduction. Bayesian statistical tests are used to assess the significant differences among algorithms (Benavoli et al., 2017).

Several surveys on distance metric learning have been proposed. Among the well-known surveys we can find the work of Yang and Jin (2006), Kulis et al. (2013), Bellet et al. (2013) and Moutafis et al. (2017). In our paper, we want to differ from these previous publications by focusing on a deeper analysis of the main concepts of distance metric learning, trying to get to its basic ideas, as well as providing an extensive software framework for metric learning purposes. Besides, we will discuss some opportunities for future work in this topic.

Our paper is organized as follows. Section 2 presents the mathematical foundations of distance metric learning, structured in the three blocks discussed previously. Section 3 introduces the distance metric problem, explains the family of distances we will work with and shows several examples and applications. Section 4 discusses all the distance metric learning algorithms chosen for this tutorial. Section 5 presents the software developed together with this paper. In this section we compare our package with the existing software and provide basic instructions for installation and use, together with links to the full documentation. Section 6 describes the experiments done to evaluate the performance of the algorithms and shows the obtained results. Finally, Sections 7 and 8 conclude the paper by summarizing the work done and indicating possible future avenues of research in this area, respectively.

2. Mathematical Background

In this section we will study three mathematical blocks that make up the foundations of distance metric learning: convex analysis, matrix analysis and information theory.

2.1 Convex Analysis

Convex analysis is a fundamental field of study for many optimization problems. This field studies the convex sets, functions and problems. Convex functions have very useful properties in optimization tasks, and allow tools to be built to solve numerous types of convex optimization problems.

We will highlight some results of convex analysis in our work. First, we will show some important geometric properties of convex sets, such as the convex projection theorem, and then we will analyze some optimization methods that will be used later.

1. `https://github.com/jlsuarezdiaz/pyDML`.

We start with the geometry of convex sets. We will work in the euclidean d-dimensional space, $\mathbb{R}^d$, where we note the dot product as $\langle \cdot, \cdot \rangle$.

2.1.1 Convex Set Results

Recall that convex sets are those for which any segment between two points in the set remains within the set, that is, a set $K \subset \mathbb{R}^d$ is convex iff $[x, y] = \{(1 - \lambda)x + \lambda y \colon \lambda \in [0, 1]\} \subset K$, for every $x, y \in K$. An important result from convex sets states that, at every point on the border of a closed set, we can setup a hyperplane so that the convex set and the hyperplane intersect only at the boundary of the set, and the whole set lies on one side of the hyperplane. Furthermore, this property characterizes the closed convex sets with non empty interiors. This result is known as the supporting hyperplane theorem and we discuss it below.

Definition 1 *Let $T \colon \mathbb{R}^d \to \mathbb{R}$ be a linear map, $\alpha \in \mathbb{R}$ and $P = \{x \in \mathbb{R}^d \colon T(x) = \alpha\}$ be an hyperplane. Associated with P, we define $P^+ = \{x \in \mathbb{R}^d \colon T(x) \geq \alpha\}$ and $P^- = \{x \in \mathbb{R}^d \colon T(x) \leq \alpha\}$.*

We say that P is a supporting hyperplane *for the set $K \subset \mathbb{R}^d$ if $P \cap \overline{K} \neq \emptyset$, and either $K \subset P^+$ or $K \subset P^-$. We refer to* supporting half-space *as the half-space that contains K, between P^+ and P^-.*

Theorem 2 (Supporting hyperplane theorem)

1. *If $K \subset \mathbb{R}^d$ is a closed convex set, then for each $x_0 \in \operatorname{Fr} K$ there is a supporting hyperplane P for K so that $x_0 \in P$.*
2. *Every proper closed convex set in $\mathbb{R}^d$ is the intersection of all its supporting half-spaces.*
3. *Let $K \subset \mathbb{R}^d$ be a closed set with non empty interior. Then, K is convex if and only if for every $x \in \operatorname{Fr} K$ there is a supporting hyperplane P for K with $x \in P$.*

Proof of this result can be found in Dacorogna (2007, chap. 2, theorem 2.7). We will use this theorem in the following results. The following property is fundamental to be able to make sense of the optimization tools shown in this paper. We will see that, given a closed convex set and a point in $\mathbb{R}^d$, we can find a nearest point to the given point in the convex set, and it is unique, that is, there is a projection for the given point onto the convex set. In other words, projections onto convex sets are well defined. We prove this result below. We will see that projections will help us to deal with constrained convex problems.

Theorem 3 (Convex projection) *Let $K \subset \mathbb{R}^d$ be a non empty closed convex set. Then, for every $x \in \mathbb{R}^d$ there is a single point $x_0 \in K$ with $d(x, K) = d(x, x_0)$, where we have defined the distance to the set K by*

$$d(x, K) = \inf\{d(x, y) \colon y \in K\}.$$

The point x_0 is called the projection *of x onto K and it is usually denoted by $P_K(x)$. The function $P_K \colon \mathbb{R}^d \to K$ given by the mapping $x \mapsto P_K(x)$ is therefore well defined and it is called the projection onto K. In addition, for each $x \in \mathbb{R}^d \setminus K$, the half-space $\{y \in \mathbb{R}^d \colon \langle x - P_K(x), y - P_K(x) \rangle \leq 0\}$ is a supporting half-space for K in $P_K(x)$.*

Proof First, we will prove the existence of a point in K in which the distance to K is achieved. In fact, this is true for every closed and not necessarily convex set. Let $x \in \mathbb{R}^d$. As K is closed, we can choose $R > 0$ so that $K \cap \overline{B}(x, R)$ is a compact and non empty set. We consider the distance to x in this set, that is, we define the map $d_x \colon K \cap \overline{B}(x, R) \to \mathbb{R}_0^+$ by $d_x(y) = d(x, y) = \|x - y\|$. d_x is continuous and it is defined over a compact set, so it attains a minimum at a point $x_0 \in K \cap \overline{B}(x, R)$.

If we now take $y \in K \cap \overline{B}(x, R)$, we get $d(x, y) = d_x(y) \geq d_x(x_0) = d(x, x_0)$. On the other hand, if we take $y \in K \setminus \overline{B}(x, R)$, we get $d(x, y) > r \geq d(x, x_0)$. We have obtained that $d(x, y) \geq d(x, x_0)$ for every $y \in K$, and therefore $d(x, K) \geq d(x, x_0)$. The remaining inequality is clear, since $x_0 \in K$, that is, x_0 is the point we were looking for.

We will see now the uniqueness of the point found. Suppose that $x_1, x_2 \in K$ verify that $d(x, x_1) = d(x, K) = d(x, x_2)$. We define x_0 as the half point in the segment $[x_1, x_2]$. We have that $x_0 \in K$, since K is convex. Let us note that

$$\langle x_1 - x_2, x - x_0 \rangle = \langle x_1 - x_2, x - \frac{1}{2}(x_1 + x_2) \rangle = \frac{1}{2} \langle x_1 - x_2, 2x - x_1 - x_2 \rangle.$$

If we substitute $x_1 - x_2 = (x - x_2) - (x - x_1)$ and $2x - x_1 - x_2 = (x - x_2) + (x - x_1)$, we obtain

$$\begin{aligned}
\langle x_1 - x_2, x - x_0 \rangle &= \frac{1}{2} \langle (x - x_2) - (x - x_1), (x - x_2) + (x - x_!) \rangle \\
&= \frac{1}{2} (\|x - x_2\|^2 - \|x - x_1\|^2) \\
&= \frac{1}{2} (d(x, K)^2 - d(x, K)^2) = 0.
\end{aligned}$$

Therefore, the vectors $x_1 - x_2$ and $x - x_0$ are orthogonal, and consequently so are $x - x_0$ y $x_0 - x_2 = (x_1 - x_2)/2$. Applying Pythagorean theorem we have

$$d(x, K)^2 = \|x - x_2\|^2 = \|x - x_0\|^2 + \|x_0 - x_2\|^2 \geq \|x - x_0\|^2 \geq d(x, K)^2,$$

that is, the equality holds in the previous inequality. In particular, we obtain that $\|x_0 - x_2\|^2 = 0$, and then $x_0 = x_2$. Since x_0 was the half point of $[x_1, x_2]$ we conclude that $x_1 = x_2$, proving the uniqueness.

Finally we will prove the last assertion in the theorem. Let $x \in \mathbb{R}^d \setminus K$ and suppose that there exists $y \in K$ with $\langle x - P_K(x), y - P_K(x) \rangle > 0$. Since K is convex, the segment $[y, P_K(x)]$ is contained in K, and therefore we have $y_t = P_K(x) + t(y - P_K(x)) \in K$, for every $t \in [0, 1]$. We define the map $f \colon [0, 1] \to \mathbb{R}$ by

$$\begin{aligned}
f(t) &= \|y_t - x\|^2 = \|P_K(x) - x + t(y - P_K(x))\|^2 \\
&= \|P_K(x) - x\|^2 + 2t \langle P_K(x) - x, y - P_K(x) \rangle + t^2 \|y - P_K(x)\|^2.
\end{aligned}$$

f is a polynomial in t, so it is differentiable, and

$$f'(0) = 2 \langle P_K(x) - x, y - P_K(x) \rangle = -2 \langle x - P_K(x), y - P_K(x) \rangle < 0.$$

Last expression implies that f is strictly decreasing in a neighborhood of 0, that is, there exists $\varepsilon > 0$ so that $\|y_t - x\|^2 < \|y_0 - x\|^2 = \|P_K(x) - x\|^2$, for $0 < t < \varepsilon$, which results in a contradiction, since $P_K(x)$ minimizes the distance to x in K and the points y_t lie on K. ■

2.1.2 Optimization Methods

In the following paragraphs we will discuss some of the optimization methods that we will use in distance metric learning algorithms. These algorithms will generally try to optimize (we will focus on minimizing without loss of generality) differentiable functions without constraints, or convex functions subject to convex constraints. For the first case, it is well known that the gradient of a differentiable function has the direction of the maximum slope in the function graph, thus by advancing small quantities in the negative gradient direction we manage to reduce the value of our objective function. This iterative method is usually called the gradient descent method. The adaptation rule for this method, for a differentiable function $f\colon \mathbb{R}^d \to \mathbb{R}$, is given by $x_{t+1} = x_t - \eta\nabla f(x_t)$, $t \in \mathbb{N} \cup \{0\}$, where η is the quantity we advance in the negative gradient direction, and it is called the *learning rate*. This value can be either constant or adapted according to the evaluations of the objective function. For the first option, the choice of a value of η that is too big or too small can lead to poor results. The second option needs to evaluate the objective function at each iteration, which can be computationally expensive.

Foundations of gradient descent are based on the following ideas. Let us consider an objective function $f\colon \mathbb{R}^d \to \mathbb{R}$, $x \in \mathbb{R}^d$ and $v \in \mathbb{R}^d \setminus \{0\}$ an arbitrary direction. We also consider the function $g\colon \mathbb{R} \to \mathbb{R}$ given by $g(\eta) = f(x + \eta v/\|v\|)$. The rate of change or directional derivative of f at x in the direction of v is given by $g'(0) = \langle \nabla f(x), v\rangle/\|v\|$. Applying Cauchy-Schwarz inequality, we have

$$-\|\nabla f(x)\| \leq \frac{1}{\|v\|}\langle \nabla f(x), v\rangle \leq \|\nabla f(x)\|,$$

and equality in the left inequality holds when $v = -\nabla f(x)$, thus obtaining the maximum descent rate. In the same way, the maximum ascent rate is achieved when $v = \nabla f(x)$.

If gradient at x is non zero and we consider the first order Taylor approximation with the points x and $x - \eta\nabla f(x)$, we have that

$$f(x - \eta\nabla f(x)) = f(x) - \eta\|\nabla f(x)\|^2 + o(\eta),$$

with $\lim_{\eta\to 0} |o(\eta)|/\eta = 0$, then there is $\varepsilon > 0$ so that if $0 < \delta < \varepsilon$, we have

$$\frac{o(\delta)}{\delta} < \|\nabla f(x)\|,$$

and therefore

$$f(x - \delta\nabla f(x)) - f(x) = \delta\left(-\|\nabla f(x)\|^2 + \frac{o(\delta)}{\delta}\right) < \delta(-\|\nabla f(x)\|^2 + \|\nabla f(x)\|^2) = 0,$$

thus $f(x - \delta\nabla f(x)) < f(x)$ for $0 < \delta < \varepsilon$, so we are guaranteed that for an accurate learning rate the gradient method performs a descent at each iteration. Let us observe that the gradient direction is not the only valid descent direction, but the above calculations are still true for any direction $v \in \mathbb{R}^d$ with $\langle \nabla f(x), v\rangle < 0$. The choice of different descent directions, even if they are not the maximum slope direction, may provide better results in certain situations.

Now we will discuss the constrained convex optimization problems. When we work with constrained problems, gradient descent method cannot be applied directly, as the gradient descent adaptation rule, $x_{t+1} = x_t - \eta\nabla f(x_t)$, does not guarantee x_{t+1} to be a feasible point, that is, a point that fulfills all the constraints. When the optimization problem is convex, the set determined by the constraints is closed and convex, so we can take projections onto this feasible set. The projected gradient method tries to fix the gradient descent problem by adding a projection onto the feasible set in the gradient descent adaptation rule, that is, if C is the feasible set, and P_C is the projection onto this set, the projected gradient adaptation rule becomes $x_{t+1} = P_C(x_t - \eta\nabla f(x_t))$. To confirm that this method is successful, we have to show that the direction $v = P_C(x - \eta\nabla f(x)) - x$ is a descent direction, which is attained, thanks to the reasons given above, if $\langle\nabla f(x), v\rangle < 0$.

We name $x_1 = x - \eta\nabla f(x)$. Then, $v = P_C(x_1) - x$. Note that $\langle\nabla f(x), v\rangle < 0 \iff \langle x_1 - x, P_C(x_1) - x\rangle = -\eta\langle\nabla f(x), v\rangle > 0$. If gradient is not null and $x_1 \in C$, we get $\langle x_1 - x, P_C(x_1) - x\rangle = \langle x_1 - x, x_1 - x\rangle = \|x_1 - x\|^2 > 0$. If $x_1 \notin C$, then the convex projection theorem (Theorem 3) ensures that the half-space $H = \{y \in \mathbb{R}^d \colon \langle x_1 - P_C(x_1), y - P_C(x_1)\rangle \leq 0\}$ contains C. In particular,

$$0 \geq \langle x_1 - P_C(x_1), x - P_C(x_1)\rangle = \langle x_1 - x, x - P_C(x_1)\rangle + \|x - P_C(x_1)\|^2.$$

Consequently, $\langle x_1 - x, P_C(x_1) - x\rangle \geq \|x - P_C(x_1)\|^2 \geq 0$. In addition, equality holds if and only if $x = P_C(x_1)$, in which case the iterative algorithm will have converged (observe that this happens when $x \in \operatorname{Fr} C$ and the gradient descent direction points out of C and orthogonally to the supporting hyperplane). Therefore, as long as the projected gradient iterations produce changes in the obtained points, an appropiate learning rate will ensure the descent in the objective function. Figure 1 visually compares the gradient descent method and the projected gradient method.

Another problem we can find when trying to optimize constrained convex problems is that we may have multiple constraints, but we only know the projection onto each single restriction, without knowing the projection onto the intersection, which makes up the feasible set. In these cases, a popular method to find a point in the intersection is the so-called *iterated projections method*, which consists of taking successive projections onto each constraint set, and repeating this procedure cyclically. We will analyze the simplest case, that is, let us suppose that we have a feasible set determined by two convex constraints. The following theorem states that, if the intersection of the sets determined by each constraint is not empty, then the sequence of iterated projections converge to a point in the intersection.

Theorem 4 (Convergence of the iterated projections method) *Let $C, D \subset \mathbb{R}^d$ be closed convex sets, and let $P_C, P_D \colon \mathbb{R}^d \to \mathbb{R}^d$ be the projections onto C and D, respectively. Suppose that $x_0 \in C$ and we build the sequences $\{x_n\}$ and $\{y_n\}$ given by $y_n = P_D(x_n)$ and $x_{n+1} = P_C(y_n)$, for each $n \in \mathbb{N} \cup \{0\}$.*

Then, if $C \cap D \neq \emptyset$, both sequences converge to a point $x^* \in C \cap D$.

Proof of this result is provided by Boyd and Dattorro (2003). It is also interesting to remark that, when the sets do not intersect, both sequences converge, as long as there are points where the distance between both sets is attained (these points will be the limits for each sequence). The extension to the general case can be made following a similar

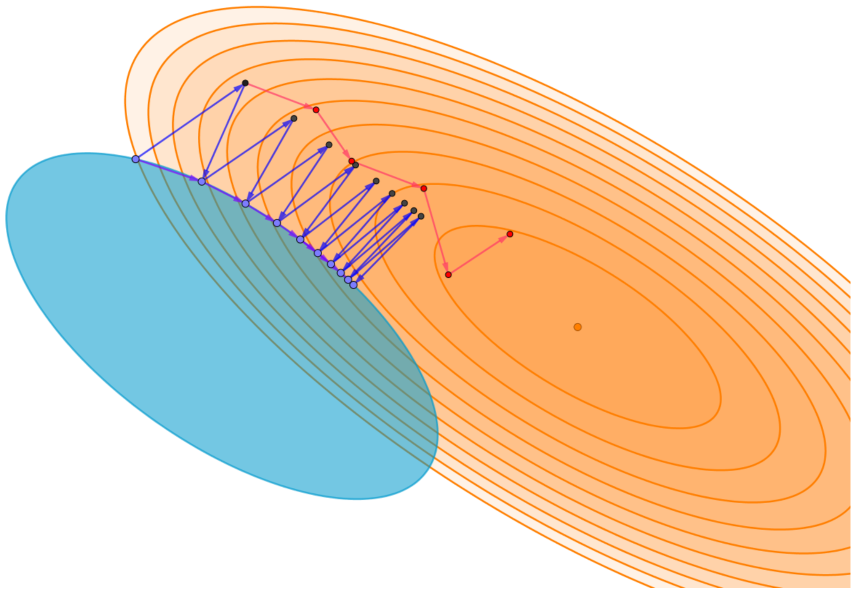

Figure 1: Orange shaded areas represent contour lines of the function $f(x, y) = 2(x + y)^2 + 2y^2$ for natural values between 0 and 10. The red path shows the behaviour of the unconstrained gradient descent method applied to f. The blue path shows the behaviour of the projected gradient descent, with the blue ellipse as the feasible set. In both cases we observe that we are obtaining descent directions.

argument, and it is discussed by Bregman (1967). That is why the general case is also called the *Bregman projections method.* Figure 2 shows a graphical example of the iterated projections method.

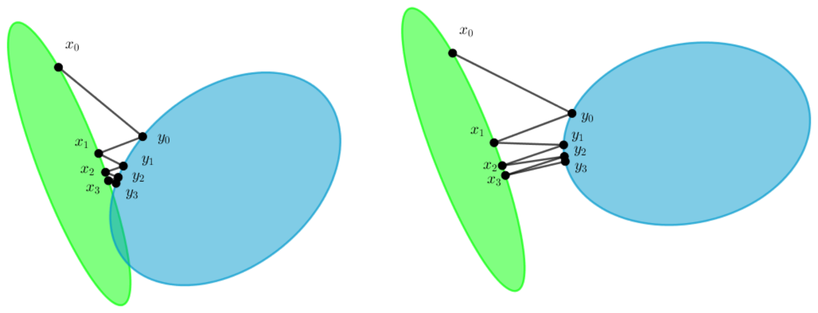

Figure 2: The iterated projections method. The second image shows how the algorithm works if the sets do not intersect.

To conclude this section, we need to make a last remark. Recall that a convex and differentiable function $f\colon \Omega \to \mathbb{R}$ defined on a convex open set verifies that $f(x) \geq \langle \nabla f(x_0), x - x_0 \rangle$, for every $x, x_0 \in \Omega$. Let $x_0 \in \Omega$ be fix. When we work with convex but non differentiable functions, there are still vectors $v \in \mathbb{R}^d$ for which $f(x) \geq f(x_0) + \langle v, x - x_0 \rangle$, for every $x \in \Omega$. This is a consequence of the supporting hyperplane theorem applied to the epigraph of f (recall that f is convex iff its epigraph is too). In this case we say that v is a *subgradient* of f at x_0 and we note it as $\partial f(x_0)$, or $\partial f(x_0)/\partial x$, if we need to specify the variable.

Subgradients and gradients have similar behaviours, although we cannot always guarantee that subgradients are descent directions. Subgradient methods work in a similar way to gradient methods, replacing the gradient in the adaptation rule by a subgradient. In subgradient methods it is useful to keep track of the best value obtained, as some subgradients may not be descent directions. In the situations we will handle, subgradient computations are easy: if f is differentiable at x_0, then $\nabla f(x_0)$ is a subgradient (in fact, this is the only subgradient at x_0); if f is a maximum of convex differentiable functions, then a subgradient at x_0 is the gradient of any of the differentiable functions that attains the maximum at x_0.

2.2 Matrix Analysis

In distance metric learning, matrices will play a key role, as they will be the structure over which distances will be defined and over which the optimization methods studied in the previous section will be applied. Within the set of all matrices, positive semidefinite matrices will be of even greater importance, so, in order to better understand the learning problems we will be dealing with, it will be necessary to delve into some of their numerous properties.

This section examines in depth the study of matrices, on a basis of the best-known results of diagonalization in linear algebra. From this basis, we will show how to give the set of matrices a Hilbert space structure, in order to be able to apply the convexity results and optimization methods from the previous chapter. In particular, we will be interested in how to obtain projections onto the set of positive semidefinite matrices. Also related to positive semidefinite matrices, we will present several results regarding decomposition that we will need in future sections. Finally, we will study some matrix optimization problems that can be solved via eigenvectors. Table 1 shows the notations we will use for matrices. We will restrict the study to the real case, since the problem we will deal with is in real variables, although many of the results we will see can be extended to the complex case.

Notation	Concept
$\mathcal{M}_{d'\times d}(\mathbb{R})$	Matrices of order $d' \times d$.
$\mathcal{M}_d(\mathbb{R})$	Square matrices of orden d.
A_{ij}	The value of the matrix A at the i-th row and j-th column.
$A_{.j}$ (resp. $A_{i.}$)	The j-th column (resp. the i-th row) of the matrix A.
$v = (v_1, \dots, v_d)$	A vector in $\mathbb{R}^d$. Vectors will be treated as column matrices.
A^T	The transpose of the matrix A.
$S_d(\mathbb{R})$	Symmetric matrices of order d.
$\mathrm{GL}_d(\mathbb{R})$	Invertible matrices of order d.
$r(A), \mathrm{tr}(A), \det(A)$	The rank, trace and determinant of the matrix A.
$O_d(\mathbb{R})$	Orthogonal matrices of order d.
$S_d(\mathbb{R})_0^+$	Positive semidefinite matrices of order d.
$S_d(\mathbb{R})^+$	Positive definite matrices of order d.
$S_d(\mathbb{R})_0^-$	Negative semidefinite matrices of order d.
$S_d(\mathbb{R})^-$	Negative definite matrices of order d.

Table 1: Matrices notations.

2.2.1 Matrices as a Hilbert Space. Projections.

Over the set of matrices we have defined a sum operation, and a matrix product, between matrices of orders $d \times r$ and $r \times n$. When working with square matrices, this sum and product give the matrix set a non-conmutative ring structure. These operations only allow us to obtain algebraic properties of matrices, but we also want to obtain geometric and topological properties. That is why we need to introduce a matrix inner product. We will introduce this inner product in the simplest way, that is, we will view matrices as vectors where we add the matrix rows one after the other, and we will consider the usual vector inner product. This matrix product is known as *Frobenius inner product.*

Definition 5 *We define the* Frobenius inner product *over the matrices space of order $d' \times d$ as the mapping $\langle \cdot, \cdot \rangle_F \colon \mathcal{M}_{d'\times d}(\mathbb{R}) \times \mathcal{M}_{d'\times d}(\mathbb{R}) \to \mathbb{R}$ given by*

$$\langle A, B \rangle_F = \sum_{i=1}^{d'} \sum_{j=1}^{d} A_{ij} B_{ij} = \mathrm{tr}(A^T B).$$

We define the Frobenius norm *over the matrices space of order $d' \times d$ as the mapping $\|\cdot\|_F \colon \mathcal{M}_{d'\times d}(\mathbb{R}) \to \mathbb{R}_0^+$ given by*

$$\|A\|_F = \sqrt{\langle A, A\rangle} = \sqrt{\sum_{i=1}^{d'}\sum_{j=1}^{d} A_{ij}^2} = \sqrt{\operatorname{tr}(A^TA)}.$$

Frobenius norm is therefore identical to the euclidean norm in $\mathbb{R}^{d'\times d}$ identifying matrices with vectors as mentioned before. Viewing this norm as a matrix norm, we have to remark that Frobenius norm is sub-multiplicative, but it is not induced by any vector norm. Some interesting properties about Frobenius norm can be deduced from the definition. They are listed below.

Proposition 6

1. *For each $A \in \mathcal{M}_{d'\times d}(\mathbb{R})$, $\|A\|_F = \|A^T\|_F$.*
2. *For each $A \in \mathcal{M}_{d'\times d}(\mathbb{R})$, $\|A\|_F = \sqrt{\operatorname{tr}(AA^T)}$*
3. *If $U \in O_d(\mathbb{R}), V \in O_{d'}(\mathbb{R})$ and $A \in \mathcal{M}_{d'\times d}(\mathbb{R})$, then $\|AU\|_F = \|VA\|_F = \|VAU\|_F = \|A\|_F$.*
4. *If $A \in S_d(\mathbb{R})$, then $\|A\|_F^2 = \sum_{i=1}^d \lambda_i^2$, where $\lambda_1, \dots, \lambda_d$ are the eigenvalues of A.*

With Frobenius inner product we can apply the convex analysis theory studied in the previous section. The positive semidefinite matrix set has a convex cone structure, that is, it is closed under non-negative linear combinations. That is why $S_d(\mathbb{R})_0^+$ is usually called the positive semidefinite cone. Under the topology induced by symmetric matrices, we can also see that $S_d(\mathbb{R})_0^+$ is closed, as it is the intersection of closed sets:

$$S_d(\mathbb{R})_0^+ = \{M \in S_d(\mathbb{R}) \colon x^TMx \geq 0 \ \forall x \in \mathbb{R}^d\} = \bigcap_{x\in\mathbb{R}^d} \{M \in S_d(\mathbb{R}) \colon x^TMx \geq 0\}.$$

So we understand, in particular, that $S_d(\mathbb{R})_0^+$ is a closed convex set over symmetric matrices, and thus we have a well-defined projection onto the positive semidefinite cone. This property is very important for many of the optimization problems we will study, since they will try to optimize functions defined over the positive semidefinite cone. Here, the projected gradient descent method will be of great use, thus constituting one of the most basic algorithms of the paradigm of semidefinite programming. We can calculate the projection onto the positive semidefinite cone explicitly, as we will see below.

Definition 7 *Let $\Sigma \in \mathcal{M}_d(\mathbb{R})$ be a diagonal matrix, $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_d)$. We define the* positive part *of Σ as $\Sigma^+ = \operatorname{diag}(\sigma_1^+, \dots, \sigma_d^+)$, where $\sigma_i^+ = \max\{\sigma_i, 0\}$. In a similar way, we define its* negative part *as $\Sigma^- = \operatorname{diag}(\sigma_1^-, \dots, \sigma_d^-)$, where $\sigma_i^- = \max\{-\sigma_i, 0\}$.*

Let $A \in S_d(\mathbb{R})$ and let $A = UDU^T$ be a spectral decomposition of A. We define the positive part *of A as $A^+ = UD^+U^T$. In a similar way we define its* negative part *as $A^- = UD^-U^T$.*

Theorem 8 (Semidefinite projection) *Let $A \in S_d(\mathbb{R})$. Then, A^+ is the projection of A onto the positive semidefinite cone.*

This result has been proven by Higham (1988), and is extended easily to project any square matrix onto the positive semidefinite cone, as we show below, although the most interesting case is that mentioned previously.

Corollary 9 *Let $A \in \mathcal{M}_d(\mathbb{R})$. Then, the projection of A onto the positive semidefinite cone is given by $((A + A^T)/2)^+$.*

2.2.2 Decomposition Theorems

The positive semidefinite cone allows many of the concepts and properties that we already know about the non negative real numbers to be generalized. For example, we can similarly define concepts as the square roots, and modules or absolute values. These concepts play an important role in elaborating numerous decomposition theorems that involve positive semidefinite matrices. All this theory will be developed in Appendix A, in order to prove a specific decomposition theorem that will motivate the ways of modeling the distance metric learning problem. The statement of this theorem is shown below.

Theorem 10 *Let $M \in S_d(\mathbb{R})_0^+$. Then,*

1. *There is a matrix $L \in \mathcal{M}_d(\mathbb{R})$ so that $M = L^T L$.*
2. *If $K \in \mathcal{M}_d(\mathbb{R})$ is any other matrix with $M = K^T K$, then $K = UL$, where $U \in O_d(\mathbb{R})$ (that is, L is unique up to isometries).*

2.2.3 Matrix Optimization Problems

To conclude the section about matrix analysis, we consider that the analysis of several specific optimization problems based on eigenvectors is necessary. These problems can be expressed as the maximization of a trace, and they do not need analytical methods, like gradient methods, to find a solution to them. It can be solved only via algebraic methods, specifically by calculating the eigenvectors of the matrices involved in the problem. We will see in subsequent sections that these problems appear in most of the dimensionality reduction algorithms. We state these problems, together with their solutions, in the lines below.

Theorem 11 *Let $d', d \in \mathbb{N}$, with $d' \leq d$. Let $A \in \mathcal{S}_d(\mathbb{R})$, and we consider the optimization problem*

$$\begin{aligned} \max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad & \operatorname{tr}\left(LAL^T\right) \\ \text{s.t.:} \quad & LL^T = I. \end{aligned}$$

The problem attains a maximum if $L = \begin{pmatrix} — & v_1 & — \\ & \dots & \\ — & v_{d'} & — \end{pmatrix}$, where $v_1, \dots, v_{d'}$ are orthonormal eigenvectors of A corresponding to its d' largest eigenvalues. In addition, the maximum value is the sum of the d' largest eigenvalues of A.

Theorem 12 *Let $d', d \in \mathbb{N}$, with $d' \leq d$. Let $A \in S_d(\mathbb{R})$ and $B \in S_d(\mathbb{R})^+$, and we consider the optimization problem*

$$\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad \operatorname{tr}\left((LBL^T)^{-1}(LAL^T)\right)$$

The problem attains a maximum if $L = \begin{pmatrix} — & v_1 & — \\ & \dots & \\ — & v_{d'} & — \end{pmatrix}$*, where $v_1, \dots, v_{d'}$ are eigenvectors of $B^{-1}A$ corresponding to its d' largest eigenvalues.*

Theorem 13 *Let $d', d \in \mathbb{N}$, with $d' \leq d$. Let $A, B \in S_d(\mathbb{R})^+$, and we consider the optimization problem*

$$\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad \operatorname{tr}\left((LBL^T)^{-1}(LAL^T) + (LAL^T)^{-1}(LBL^T)\right)$$

The problem attains a maximum if $L = \begin{pmatrix} — & v_1 & — \\ & \dots & \\ — & v_{d'} & — \end{pmatrix}$*, where $v_1, \dots, v_{d'}$ are the d' eigenvectors of $B^{-1}A$ with the highest values for the expression $\lambda_i + 1/\lambda_i$, where λ_i is the eigenvalue associated with v_i.*

These theorems can be proven using tools such as the Rayleigh quotient and the Courant-Fischer theorem and its consequences. This theory will be developed in Appendix B.

2.3 Information Theory

Information theory is a branch of mathematics and computer theory, with the purpose of establishing a rigurous measure to quantify the information and disorder contained in a communication message. It was developed with the aim of finding limits in signal processing operations such as compression, storage and communication. Today, its applications extend to most fields of science and engineering.

Many concepts associated with information theory have been defined, such as entropy, which measures the amount of uncertainty or information expected in an event, mutual information, which measures the amount of information that one random variable contains about another random variable, or relative entropy, which is a way of measuring the closeness between different random variables. We will focus on the relative entropy, and the concepts derived from it. To do this, we will first define the concept of divergence. Divergence is a magnitude to measure the closeness between certain objects in a set. We should not confuse divergences with distances (we will revisit this concept in Section 3.1), because the magnitudes we will consider may not verify some of the properties required for distances, such as symmetry or triangle inequality.

Definition 14 *Let X be a set. A map $D(\cdot\|\cdot) \colon X \times X \to \mathbb{R}$ is said to be a* divergence *if it verifies the following properties:*

1. *Non negativity: $D(x\|y) \geq 0$, for every $x, y \in X$.*

2. *Coincidence:* $D(x\|y) = 0$ *if, and only if,* $x = y$.

We will use divergences to measure the closeness between probability distributions. The divergences we will use will be presented in the following paragraphs.

Definition 15 *Let* $(\Omega, \mathcal{A}, P)$ *be a probability space and* $X \colon \Omega \to \mathbb{R}$ *be a random variable, discrete or continuous, in that space. Suppose that* p *is the corresponding probability mass function or density function. Suppose that* q *is another probability mass function or density function. Then, we define the* relative entropy *or the* Kullback-Leibler *divergence between* p *and* q*, as*

$$\mathrm{KL}(p\|q) = \mathbb{E}_p\left[\log\frac{p(X)}{q(X)}\right],$$

as long as such expectation exists. For the discrete case, if p *and* q *are valued over the same points, we have*

$$\mathrm{KL}(p\|q) = \sum_{x\in X(\Omega)} p(x)\log\frac{p(x)}{q(x)},$$

and for the continuous case, as long as the absolute integral is finite, we have

$$\mathrm{KL}(p\|q) = \int_{-\infty}^{+\infty} p(x)\log\frac{p(x)}{q(x)}\ dx.$$

For continuity reasons, we assume that $0\log(0/0) = 0$.

The first step is to check that, indeed, Kullback-Leibler divergence is a divergence. This result is known as the information inequality.

Theorem 16 (Information inequality) *Kullback-Leibler divergence is a divergence, that is,* $\mathrm{KL}(p\|q) \geq 0$ *and the equality holds if, and only if,* $p(x) = q(x)$ *a.e. in* $X(\Omega)$ *(the equality is at every point in the discrete case).*

Proof This result is an immediate consequence of Jensen's inequality (see Rudin, 1987, chap. 3, theorem 3.3) applied to the $-\log$ function, which is strictly convex. We have

$$\begin{aligned}\mathrm{KL}(p\|q) &= \mathbb{E}_p\left[\log\frac{p(X)}{q(X)}\right] = \mathbb{E}_p\left[-\log\frac{q(X)}{p(X)}\right]\\ &\geq -\log\mathbb{E}_p\left[\frac{q(X)}{p(X)}\right] = -\log\int p(x)\frac{q(x)}{p(x)}\ dx\\ &= -\log\int q(x)\ dx = -\log 1 = 0.\end{aligned}$$

The proof for the discrete case is similar. In addition, the strict convexity implies that equality holds iff p/q is constant a.e., iff $p = q$ a.e., since they are probability density functions or mass functions. And, as in the discrete case p and q are valued over sets with no null probabilities, we have equality at every point. ■

As we have already mentioned, Kullback-Leibler divergence is useful to measure closeness between probability distributions and can be used to bring the distributions closer. However, it is not all that useful to put the distributions away, since, as Kullback-Leibler divergence is not symmetric, the values of $\mathrm{KL}(p\|q)$ and $\mathrm{KL}(q\|p)$ may differ significantly when p and q are not near. That is why it is sometimes helpful to work with a symmetrization of the Kullback-Leibler divergence known as the Jeffrey divergence.

Definition 17 *The* Jeffrey divergence *between two probability distributions p and q for which $\mathrm{KL}(p\|q)$ and $\mathrm{KL}(q\|p)$ exist is defined by*

$$\mathrm{JF}(p\|q) = \mathrm{KL}(p\|q) + \mathrm{KL}(q\|p).$$

In the discrete case we have

$$\mathrm{JF}(p\|q) = \sum_{x \in X(\Omega)} (p(x) - q(x))(\log p(x) - \log q(x)).$$

And, for the continuous case,

$$\mathrm{JF}(p\|q) = \int_{-\infty}^{\infty} (p(x) - q(x))(\log p(x) - \log q(x))\ dx.$$

It is clear that Jeffrey divergence is a divergence, as a consequence of the information inequality, and it is also symmetric. Observe that both divergences are functions only of the probability distributions, that is, they only depend on the values set on the distributions. This fact allows divergence to be extended to random vectors, as long as we know its probability density functions or mass functions.

A case of special interest in the algorithms we will discuss in subsequent sections is the calculation of divergences between multivariate gaussian distributions. Recall that, if $\mu \in \mathbb{R}^d$ and $\Sigma \in S_d(\mathbb{R})^+$, a random vector $X = (X_1, \dots, X_d)$ follows a multivariate gaussian distribution with mean μ and covariance Σ, if it has the following probability density function:

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right).$$

It is well-known that $\mathbb{E}[X] = \mu$ and $\mathrm{Cov}(X) = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^T] = \Sigma$, thus gaussian distributions are completely defined by its mean and covariance. We want to establish an easy way to compute the calculation of divergences between gaussian distributions. To do this, we will find relationships between the studied divergences and matrix divergences. Matrix divergences are an alternative to the Frobenius norm for measuring the closeness between matrices. We are interested in the ones known as Bregman divergences.

Definition 18 *Let $K \subset \mathcal{M}_d(\mathbb{R})$ be an open convex set, and $\phi\colon K \to \mathbb{R}$ a strictly convex and differentiable function. The* Bregman divergence *corresponding to ϕ is the map $D_\phi(\cdot\|\cdot)\colon K \times K \to \mathbb{R}$ given by*

$$D_\phi(A\|B) = \phi(A) - \phi(B) - \mathrm{tr}(\nabla\phi(B)^T(A - B)).$$

Effectively, Bregman divergences are also divergences, as we can write the expression above as $D_\phi(A\|B) = \phi(A) - \phi(B) - \langle \nabla\phi(B), A - B\rangle_F$, which is known to be non negative when ϕ is strictly convex, and to take the zero value if and only if $A = B$. In our situation, we are interested in choosing the *log-det* function to construct a Bregman divergence, that is, the function $\phi_{ld}\colon S_d(\mathbb{R})^+ \to \mathbb{R}$ given by

$$\phi_{ld}(M) = -\log\det(M).$$

This function is known to be strictly convex and its gradient is $\nabla f(M) = M^{-1}$, for each M in $S_d(\mathbb{R})^+$ (see Boyd and Vandenberghe, 2004, sec. 3.1.5), hence we can construct the known as *log-det divergence* through the expression

$$D_{ld}(A\|B) = \log\det(B) - \log\det(A) - \operatorname{tr}(B^{-1}(A - B)) = \operatorname{tr}(AB^{-1}) - \log\det(AB^{-1}) - d.$$

Once defined the log-det divergence, we are able to express the Kullback-Leibler and Jeffrey divergences between gaussian distributions in terms of this new matrix divergence.

Theorem 19 *Kullback-Leibler divergence between two multivariate gaussian distributions defined by the probability density functions $p_1(x|\mu_1, \Sigma_1)$ and $p_2(x|\mu_2, \Sigma_2)$, with $\mu_1, \mu_2 \in \mathbb{R}^d$ and $\Sigma_1, \Sigma_2 \in S_d(\mathbb{R})^+$, verifies that*

$$\mathrm{KL}(p_1\|p_2) = \frac{1}{2}D_{ld}(\Sigma_1\|\Sigma_2) + \frac{1}{2}\|\mu_1 - \mu_2\|^2_{\Sigma_1^{-1}},$$

where $\|\cdot\|_\Sigma$ denotes the norm defined by the positive definite matrix Σ, that is, $\|v\|_\Sigma = \sqrt{v^T\Sigma v}$, for every $v \in \mathbb{R}^d$.

Proof of this result can be found in Davis and Dhillon (2007, sec. 3.1). A simpler version of this theorem can be stated immediately, when we consider equal-mean gaussian distributions.

Corollary 20 *Kullback-Leibler divergence between two multivariate gaussian distributions defined by the probability density functions p_1 and p_2 with equal means and covariances Σ_1 and Σ_2, verifies that*

$$\mathrm{KL}(p_1\|p_2) = \frac{1}{2}D_{ld}(\Sigma_1\|\Sigma_2).$$

Using these results, we can also express the Jeffrey divergence between gaussian distributions in terms of its mean vectors and covariance matrices. The following expressions can be easily deduced from the theorems above. For more details, see also Nguyen et al. (2017, App. B).

Corollary 21 *Jeffrey divergence between two multivariate gaussian distributions defined by the probability density functions $p_1(x|\mu_1, \Sigma_1)$ and $p_2(x|\mu_2, \Sigma_2)$ with $\mu_1, \mu_2 \in \mathbb{R}^d$ and $\Sigma_1, \Sigma_2 \in S_d(\mathbb{R})^+$, verifies that*

$$\mathrm{JF}(p_1\|p_2) = \frac{1}{2}\operatorname{tr}(\Sigma_1\Sigma_2^{-1} + \Sigma_1^{-1}\Sigma_2) - d + \frac{1}{2}\|\mu_1 - \mu_2\|^2_{\Sigma_1^{-1}+\Sigma_2^{-1}}.$$

Corollary 22 *Jeffrey divergence between two multivariate gaussian distributions defined by the probability density functions p_1 and p_2 with equal means and covariances Σ_1 and Σ_2, verifies that*

$$\mathrm{JF}(p_1\|p_2) = \frac{1}{2}\operatorname{tr}(\Sigma_1\Sigma_2^{-1} + \Sigma_1^{-1}\Sigma_2) - d.$$

3. Distance Metric Learning

In this section we will introduce the distance metric learning problem. To begin with, we will remember the concept of distance, with special emphasis on those distances that will allow us to model our problem. Next, we will describe the distance metric learning problem, and we will finish by showing some of its applications.

3.1 Mahalanobis Distances

We will start by reviewing the concept of distance and some of its properties.

Definition 23 *Let X be a non empty set. A* distance *or* metric *over X is a map $d\colon X \times X \to \mathbb{R}$ that satisfies the following properties:*

1. *Coincidence: $d(x,y) = 0 \iff x = y$, for every $x, y \in X$.*
2. *Symmetry: $d(x,y) = d(y,x)$, for every $x, y \in X$.*
3. *Triangle inequality: $d(x,z) \leq d(x,y) + d(y,z)$, for every $x, y, z \in X$.*

The ordered pair (X, d) is called a metric space*.*

The coincidence property stated above will not be important for us. That is why we will also consider mappings known as *pseudodistances*, which demand only that $d(x,x) = 0$, instead of the coincidence property. In fact, pseudodistances are very related with dimensionality reduction, which is an important application of distance metric learning. From now on, when we talk about of distances, we will be considering proper distances as well as pseudodistances.

Remark 24 *As an immediate consequence of the definition, we have the following additional properties about distances:*

4. *Non negativity: $d(x,y) \geq 0$ for every $x, y \in X$.*
5. *Reverse triangle inequality: $|d(x,y) - d(y,z)| \leq d(x,z)$ for every $x, y, z \in X$.*
6. *Generalized triangle inequality: $d(x_1, x_n) \leq \sum_{i=1}^{n-1} d(x_i, x_{i+1})$ for $x_1, \dots, x_n \in X$.*

When we work in the d-dimensional euclidean space we find a family of distances very useful in the computing field. These distances are parameterized by positive semidefinite distances and are known as *Mahalanobis distances.*

Definition 25 *Let $d \in \mathbb{N}$ and $M \in S_d(\mathbb{R})_0^+$. The* Mahalanobis distance *corresponding to the matrix M is the map $d_M\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ given by*

$$d_M(x,y) = \sqrt{(x-y)^T M (x-y)}, \quad x, y \in \mathbb{R}^d.$$

Mahalanobis distances come from the (semi-)dot products in $\mathbb{R}^d$ defined by the positive semidefinite matrix M. When M is full-rank, Mahalanobis distances are proper distances. Otherwise, they are pseudodistances. Observe that the euclidean usual distance is a particular example of a Mahalanobis distance, when M is the identity matrix I. Mahalanobis distances have additional properties specific to distances over normed spaces.

7. Homogeneousness: $d(ax, ay) = |a|d(x, y)$, for $a \in \mathbb{R}$, and $x, y \in \mathbb{R}^d$.

8. Translation invariance: $d(x, y) = d(x + z, y + z)$, for $x, y, z \in \mathbb{R}^d$.

Sometimes the term "Mahalanobis distance" is used to describe the squared distances of the form $d_M^2(x, y) = (x-y)^T M(x-y)$. In the area of computing, it is much more efficient to work with d_M^2 rather than with d_M, as this avoids the calculation of square roots. Although d_M^2 is not really a distance, it keeps the most useful properties of d_M from the distance metric learning perspective, as we will see, such as the greater or lesser closeness between different pairs of points. That is why the use of the term "Mahalanobis distance" for both d_M and d_M^2 is quite widespread.

To conclude this part, we return to the issue of dimensionality reduction that we mentioned when introducing the concept of pseudodistance. When we work with a pseudodistance σ over a set X, it is possible to define an equivalence relationship given by $x \sim y$ if and only if $\sigma(x, y) = 0$, for each $x, y \in X$. With this relationship we can consider the quotient space $X/_\sim$, and the map $\hat{\sigma}\colon X/_\sim \times X/_\sim \to \mathbb{R}$ given by $\hat{\sigma}([x], [y]) = \sigma(x, y)$, for each $[x], [y] \in X/_\sim$. This map is well defined and is a distance over the quotient space. When σ is a Mahalanobis distance over $\mathbb{R}^d$, with rank $d' < d$, then the previous quotient space becomes a vector space isomorphic to $\mathbb{R}^{d'}$, and the distance $\hat{\sigma}$ is a full-rank Mahalanobis distance over $\mathbb{R}^{d'}$. That is why, when we have a Mahalanobis pseudodistance on $\mathbb{R}^d$, we can view this as a proper Mahalanobis distance over a lower dimensional space, hence we have obtained a dimensionality reduction.

3.2 Problem Description

One of the most important components in many human cognitive processes is the ability to detect similarities between different objects. This ability has been taken to the field of machine learning by designing algorithms that learn from a dataset according to the similarities between those data.

To measure the similarity between data, it is necessary to introduce a distance, which allows us to establish a measure whereby it is possible to determine when a pair of samples is more similar than another pair of samples. However, there is an infinite number of distances we can work with, and not all of them will adapt properly to our data. Therefore, the choice of an adequate distance is a crucial element in this type of algorithm. The search for an appropiate distance is the task that is carried out in distance metric learning.

Distance metric learning is a machine learning discipline with the purpose of learning distances from a dataset. In its most general version, a dataset $\mathcal{X} = \{x_1, \dots, x_N\}$ is available, on which certain similarity measures between different pairs or triplets of data are collected. These similarities are determined by the sets

$$\begin{aligned}
S &= \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \colon x_i \text{ and } x_j \text{ are similar.}\},\\
D &= \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \colon x_i \text{ and } x_j \text{ are not similar.}\},\\
R &= \{(x_i, x_j, x_l) \in \mathcal{X} \times \mathcal{X} \times \mathcal{X} \colon x_i \text{ is more similar to } x_j \text{ than to } x_l.\}.
\end{aligned}$$

With these data and similarity constraints, the problem to be solved consists in finding, after establishing a family of distances $\mathcal{D}$, those distances that best adapt to the criteria specified by the similarity constraints. To do this, a certain loss function ℓ is set, and the distances to seek will be those that solve the optimization problem

$$\min_{d \in \mathcal{D}} \ell(d, S, D, R).$$

When we focus on supervised learning, in addition to dataset $\mathcal{X}$ we have a list of labels $y_1, \dots, y_N$ corresponding to each sample in $\mathcal{X}$. The general formulation of the distance metric learning problem is easily adapted to this new situation, just by considering the sets S and D as

$$S = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \colon y_i = y_j\},$$
$$D = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \colon y_i \neq y_j\}.$$

In addition, the set R may be also available by defining triplets (x_i, x_j, x_l) where in general $y_i = y_j \neq y_l$, and also verifying certain conditions on the distance between x_i and x_j, as opposed to the distance between x_i and x_l. This is the case, for example, for impostors in the LMNN algorithm (see Section 4.2.1 and Weinberger and Saul, 2009). In any case, labels have all the necessary information in the field of supervised distance metric learning. From now on we will focus on this kind of problem.

Furthermore, focusing on the nature of the dataset, practically all of the distance metric learning theory is developed for numerical data, due in part to the richness of the distances available to this kind of sets, and their ability to be parameterized computationally, and in part to the fact that nominal data can be converted to binary numerical variables, or ordinal variables, with an appropiate preprocessing. For this reason, from now on, we will focus on supervised learning problems with numerical datasets.

We will suppose then that $\mathcal{X} \subset \mathbb{R}^d$. As we saw in the previous section, for finite-dimensional vector spaces we have the family of Mahalanobis distances, $\mathcal{D} = \{d_M \colon M \in S_d(\mathbb{R})_0^+\}$. With this family, we have at our disposal all the distances associated with dot products in $\mathbb{R}^d$ (and in lower dimensions). In addition, this family is determined by the set of positive semidefinite matrices, and therefore, we can use these matrices, which we will call *metric matrices*, to parameterize distances. In this way, the general problem adapted to supervised learning with Mahalanobis distances can be rewritten as

$$\min_{M \in S_d(\mathbb{R})_0^+} \ell(d_M, (x_1, y_1), \dots, (x_N, y_N)).$$

However this is not the only way to parameterize this type of problem. We know, from Theorem 10, that if $M \in S_d(\mathbb{R})_0^+$, then there exists a matrix $L \in \mathcal{M}_d(\mathbb{R})$ so that $M = L^T L$, and this matrix is unique except for an isometry. So then we get

$$d_M^2(x, y) = (x - y)^T M (x - y) = (x - y)^T L^T L (x - y) = (L(x - y))^T (L(x - y)) = \|L(x - y)\|_2^2.$$

Therefore, we can also parameterize Mahalanobis distances through any matrix, although in this case the interpretation is different. When we learn distances through positive semidefinite matrices we are learning a new metric over $\mathbb{R}^d$. When we learn distances with the previous L matrices, we are learning a linear map (given by $x \mapsto Lx$) that transforms the data in the space, and the corresponding distance is the usual euclidean distance after projecting the data onto the new space through the linear map. Both approaches are equivalent thanks to Theorem 10.

In relation to dimensionality, it is important to note that, when the learned metric M is not full-rank, we are actually learning a distance over a space of lower dimension (as we mentioned in the previous section), which allows us to reduce the dimensionality of our dataset. The same occurs when we learn linear maps that are not full-rank. We can extend this case and opt to learn directly linear maps defined by $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$, with $d' < d$. In this way, we ensure that data are directly projected into a space of dimension no greater than d'.

Both learning the metric matrix M and learning the linear transformation L, are useful approaches to model distance metric learning problems, each one with its advantages and disadvantages. For example, parameterizations via M usually lead to convex optimization problems. In contrast, convexity in problems parameterized by L is not so easy to achieve. On the other hand, parameterizations through L make it possible to learn projections directly onto lower dimensional spaces, while dimensional constraints for problems parameterized by M are not so easy to achieve. Let us examine these differences with simple examples.

Example 1 *Many of the functions we will want to optimize will depend on the squared distance defined by the metric M or by the linear transformation L, that is, either they will have terms of the form $\|v\|_M^2 = v^T M v$, or of the form $\|v\|_L^2 = \|Lv\|_2^2$. Both the maps $M \mapsto \|v\|_M^2$ and $L \mapsto \|v\|_L^2$ are convex (the first is actually affine). However, if we want to substract terms in this way, we lose convexity in L, because the mapping $L \mapsto -\|v\|_L^2$ is no longer convex. In contrast, the mapping $M \mapsto -\|v\|_M^2$ is still affine and, therefore, convex.*

Example 2 *Rank constraints are not convex, and therefore we may not dispose of a projection onto the set corresponding to those constraints, unless we learn the mapping (parameterized by L) directly to the space with the desired dimension, as explained before. For example, if we consider the set $C = \{M \in S_2(\mathbb{R})_0^+ \colon r(A) \leq 1\}$, we get $A = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \in C$ and $B = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \in C$. However, $(1-\lambda)A + \lambda B = I \notin C$, for $\lambda = 1/2$.*

3.3 Some Applications

This section describes some of the main applications of distance metric learning, illustrated with several examples.

- **Improve the performance of distance-based classifiers.** This is one of the main purposes of distance metric learning. Through such learning, a distance that fits well with the dataset and the classifier can be found, improving the performance of the classifier (Weinberger and Saul, 2009; Goldberger et al., 2005). An example is shown in Figure 3.

- **Dimensionality reduction.** As we have already commented, learning a low-rank metric implies a dimensionality reduction on the dataset we are working with. This

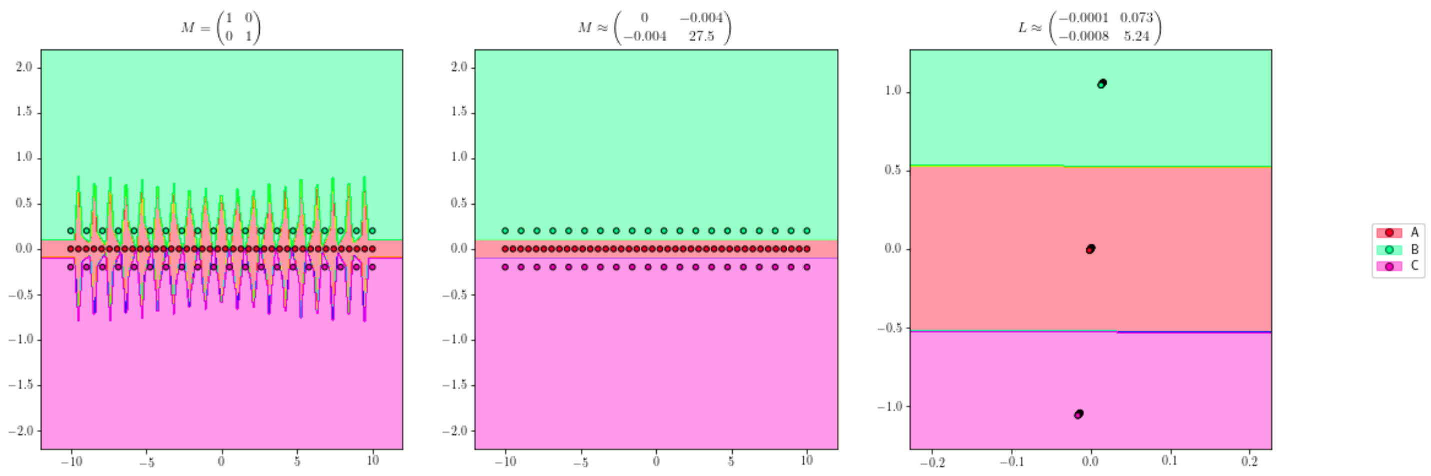

Figure 3: Suppose we have a dataset in the plane, where data can belong to three different classes, whose regions are defined by parallel lines. Suppose that we want to classify new samples using the one nearest neighbor classifier. If we use euclidean distance, we would obtain the classification regions shown in the image on the left, because there is a greater separation between each sample in class B and class C than there is between the regions. However, if we learn an adequate distance and try to classify with the nearest neighbor classifier again, we obtain much more effective classification regions, as shown in the center image. Finally, as we have seen, learning a metric is equivalent to learning a linear map and to use euclidean distance in the transformed space. This is shown in the right figure. We can also observe that data are being projected, except for precision errors, onto a line, thus we are also reducing the dimensionality of the dataset.

dimensionality reduction provides numerous advantages, such as a reduction in the computational cost, both in space and time, of the algorithms that will be used later, or the removal of the possible noise introduced when picking up the data. In addition, some distance-based classifiers are exposed to a problem called *curse of dimensionality* (see, for example, Shalev-Shwartz and Ben-David, 2014, sec. 19.2.2). By reducing the dimension of the dataset, this problem also becomes less serious. Finally, if deemed necessary, projections onto dimension 1, 2 and 3 would allow us to obtain visual representations of our data, as shown in Figure 4. In general, many real-world problems arise with a high dimensionality, and need a dimensionality reduction to be handled properly (Van Der Maaten et al., 2009).

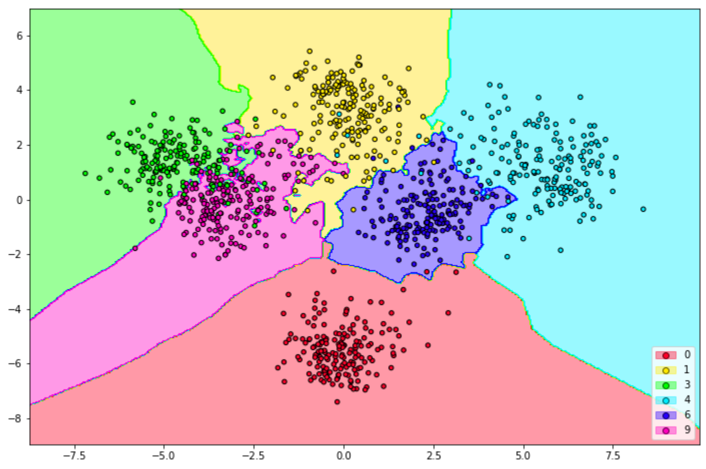

Figure 4: 'Digits' dataset consists of 1797 examples. Each of them consists of a vector with 64 attributes, representing intensity values on an 8x8 image. The examples belong to 10 different classes, each of them representing the numbers from 0 to 9. By learning an appropiate transformation we are able to project most classes on the plane, so that we perceive clearly differentiated regions associated with each of the classes.

- **Axes selection and data rearrangement.** Closely related to dimensionality reduction, this application is a result of algorithms that learn transformations which allow the coordinate axes to be moved (or selected according to the dimension), so that in the new coordinate system the vectors concentrate certain measures of information on their first components (Kokiopoulou et al., 2011). An example is shown in Figure 5.

- **Improve the performance of clustering algorithms.** Many of the clustering algorithms use a distance to measure the closeness between data, and thus establish the clusters so that data in the same cluster are considered close for that distance. Sometimes, although we do not know the ideal groupings of the data or the number of clusters to establish, we can know that certain pairs of points must be in the same

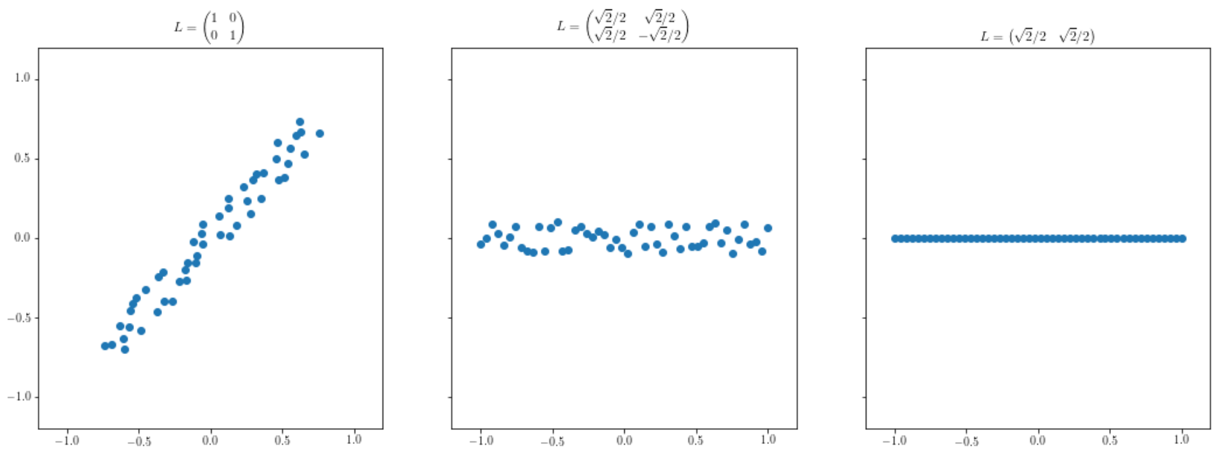

Figure 5: The dataset in the left figure seems to concentrate most of its information on the diagonal line that joins the lower left and upper right corners. By learning an appropiate transformation, we can get that direction to fall on the horizontal axis, as shown in the center image. In this way, the first coordinate of the vectors in this new basis concentrates a large part of the variability of the vector. In addition, it seems reasonable to think that the values introduced by the vertical coordinate can be due to noise, so we can even keep only the first component, as shown in the right image.

cluster and that other specific pairs must be in different clusters (Xing et al., 2003). This happens in numerous problems, for example, when clustering web documents (Aggarwal et al., 2012). These documents have a lot of additional information, such as links between documents, which can be included as similarity constraints.

- **Semi-supervised learning.** Semi-supervised learning is a learning model in which there is one set of labeled data and another set (generally much larger) of unlabeled data. Both datasets are intended to learn a model that allows new data to be labeled. Semi-supervised learning arises from the fact that in many situations collecting unlabeled data is relatively straightforward, but assigning labels can require a supervisor to assign them manually, which may not be feasible. In contrast, when a lot of unlabeled data is used along with a small amount of labeled data, it is possible to considerably improve learning outcomes, as exemplified in Figure 6. Many of these techniques consist of constructing a graph with weighted edges from the data, where the value of the edges depends on the distances between the data. From this graph we try to infer the labels of the whole dataset, using different propagation algorithms (Zhu and Ghahramani, 2002). In the construction of the graph, the choice of a suitable distance is important, thus distance metric learning comes into play (Dhillon et al., 2010).

From the applications we have seen, we can conclude that distance metric learning can be viewed as a preprocessing step for many distance-based learning algorithms. The algorithms analyzed in our work focus on the first three applications of the above enumeration.

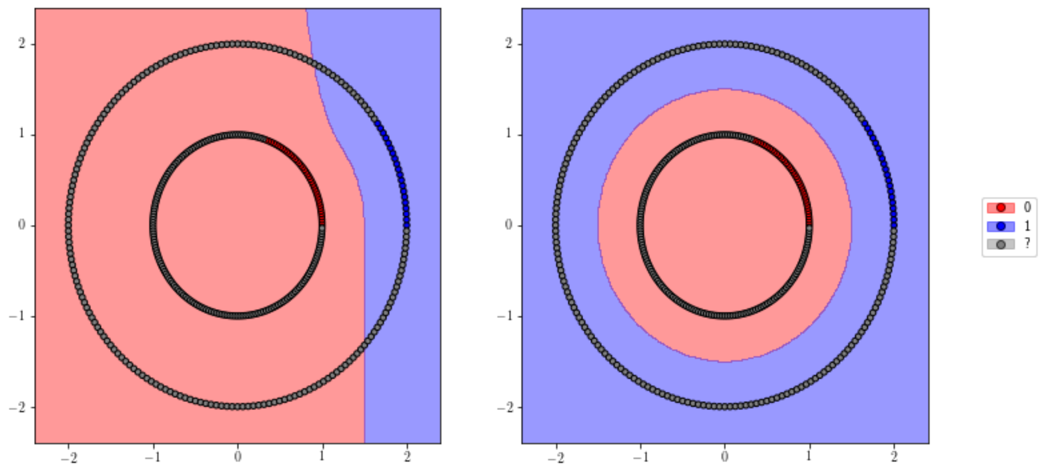

Figure 6: Learning with only supervised information (left) versus learning with all unsupervised information (right).

4. Algorithms for Distance Metric Learning

This section describes some of the most popular techniques currently being used in supervised distance metric learning. We also add a review of the principal component analysis, although not supervised, because of its importance for other distance metric learning algorithms. Some of these techniques, such as PCA or LDA (Cunningham and Ghahramani, 2015), are statistical procedures developed over the last century, which are still of great relevance in many problems nowadays. Other more recent proposals are in the state of the art, as is the case of NCMML (Mensink et al., 2012) or DMLMJ (Nguyen et al., 2017), among others. Several of the most popular classic distance metric learning algorithms, such as LMNN (Weinberger and Saul, 2009) or NCA (Goldberger et al., 2005), have also been included.

The analyzed techniques are grouped into six subsections. Each of these subsections describes algorithms that share the main purpose, although the purposes described in each section are not exclusive. In the first section we will study the techniques oriented specifically to dimensionality reduction. Next, the techniques with the purpose of learning distances that improve the nearest neighbors classifiers will be developed (Section 4.2), followed by those techniques that aim to improve classifiers based on centroids (Section 4.3). The fourth subsection includes methods based on the information theory concepts studied in Section 2.3. Subsequently, several distance metric learning mechanisms with less specific goals are described (Section 4.5). Finally, kernel-based versions of some of the above algorithms are analyzed, to be able to work in high-dimensionality spaces (Section 4.6).

For each of the techniques we will analyze the problem they try to solve or optimize, the mathematical formulations of those problems and the algorithms proposed to solve them.

4.1 Dimensionality Reduction Techniques

Dimensionality reduction techniques try to learn a distance by searching for a linear transformation from the dataset space to a lower dimensional euclidean space. These kinds of algorithms share many features. For instance, they are usually efficient and their execution involves the calculation of eigenvectors. It is important to point out that there are other non-linear or unsupervised dimensionality reduction techniques (Lee and Verleysen, 2007), but they are beyond the scope of this paper (with the exception of kernel versions in Section 4.6). The algorithms we will describe are PCA (Jolliffe, 2002), LDA (Fisher, 1936) and ANMM (Wang and Zhang, 2007).

4.1.1 PCA

PCA (*principal component analysisis*) (Jolliffe, 2002) is one of the most popular dimensionality reduction techniques in unsupervised distance metric learning. Although PCA is an unsupervised learning algorithm, it is necessary to talk about it in our work, firstly because of its great relevance, and more particularly, because when a supervised distance metric learning algorithm does not allow a dimensionality reduction, PCA can be first applied to the data in order to be able to use the algorithm later in the lower dimensional space.

Principal component analysis can be understood from two different points of view, which end up leading to the same optimization problem. The first of these approaches consists of finding two linear transformations, one that compresses the data to a smaller space, and another that decompresses them in the original space, so that in the process of compression and decompression the minimum information is lost.

Let us focus on this first approach. Suppose we have the dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, and fix $0 < d' < d$. Let us also assume that data are centered, that is, that the mean of the dataset is zero. If it is not the case, it is enough to apply previously to the data the transformation $x \mapsto x - \mu$, where $\mu = \sum x_i/N$ is the dataset mean. We are looking for a compression matrix $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$, and a decompression matrix $U \in \mathcal{M}_{d \times d'}(\mathbb{R})$, so that, after compressing and decompressing each data the squares of the euclidean distances to the original data are minimal. In other words, the problem we are trying to solve is

$$\min_{\substack{L \in \mathcal{M}_{d' \times d}(\mathbb{R}) \\ U \in \mathcal{M}_{d \times d'}(\mathbb{R})}} \sum_{i=1}^{N} \|x_i - ULx_i\|_2^2. \tag{1}$$

To find a solution to this problem, first of all we will see that U and L matrices have to be related in a very particular way.

Lemma 26 *If (U, L) is a solution of the problem given in Eq. 1, then $LL^T = I$ (in $\mathbb{R}^{d'}$) and $U = L^T$.*

Proof We fix $U \in \mathcal{M}_{d \times d'}(\mathbb{R})$ and $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$. We can assume that both U and L are full-rank, otherwise the rank of UL is lower than d'. Note that in that case, it is always possible to extend U and L matrices to full-rank matrices (by replacing linear combinations in the columns by linear independent vectors as long as the dimension allows it) so that the subspace generated extends the one generated by UL, and in such a case, the error obtained in Eq. 1 for the extension will be, at most, the error obtained for U and L.

We consider the linear map $x \mapsto ULx$. The image of this map, $R = \{ULx \colon x \in \mathbb{R}^d\}$, is a vector subspace of $\mathbb{R}^d$ of dimension d'. Let $\{u_1, \dots, u_{d'}\}$ be an orthonormal basis of R, and let $V \in \mathcal{M}_{d' \times d}(\mathbb{R})$ the matrix that has, by rows, the vectors $u_1, \dots, u_{d'}$. It is verified then that the image of V has dimension d' and that $VV^T = I$. In addition, if we consider V^T as a linear map, we see that its image is R (since $V^T e_i = u_i, i = 1, \dots, d'$, where $\{e_1, \dots, e_{d'}\}$ is the canonical basis of $\mathbb{R}^{d'}$).

Therefore, every vector of R can be written as $V^T y$, with $y \in \mathbb{R}^{d'}$. Given $x \in \mathbb{R}^d, y \in \mathbb{R}^{d'}$, we have

$$\begin{aligned}
\|x - V^T y\|_2^2 &= \langle x - V^T y, x - V^T y\rangle \\
&= \|x\|^2 - 2\langle x, V^T y\rangle + \|V^T y\|^2 \\
&= \|x\|^2 - 2\langle y, Vx\rangle + y^T V V^T y \\
&= \|x\|^2 - 2\langle y, Vx\rangle + y^T y \\
&= \|x\|^2 + \|y\|^2 - 2\langle y, Vx\rangle.
\end{aligned}$$

If we calculate the gradient with respect to y from the last previous expression, we obtain $\nabla_y \|x - V^T y\|_2^2 = 2y - 2Vx$, which, by equating to zero, allows us to obtain a single critical point, $y = Vx$. The convexity of this function (it is the composition of the euclidean norm with an affine map) assures us that this critical point is a global minimum. Therefore, this tells us that, for each $x \in \mathbb{R}^d$, the distance to x in the set R achieves its minimum at the point $V^T V x$. In particular, for the dataset $\mathcal{X}$ we conclude that

$$\sum_{i=1}^{N} \|x_i - ULx_i\|_2^2 \geq \sum_{i=1}^{N} \|x_i - V^T V x_i\|_2^2.$$

Since U and L were fixed, we can find a matrix V with these properties for any U and L in the conditions of the problem, which concludes the proof. ■

The above lemma allows us to reformulate our problem in terms of only the matrix L,

$$\min_{\substack{L \in \mathcal{M}_{d' \times d}(\mathbb{R}) \\ LL^T = I}} \quad \sum_{i=1}^{N} \|x_i - L^T L x_i\|_2^2. \tag{2}$$

Let us note now that, for $x \in \mathbb{R}^d$ and $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$, it is verified that

$$\begin{aligned}
\|x - L^T L x\|_2^2 &= \langle x - L^T L x, x - L^T L x\rangle \\
&= \|x\|^2 - 2\langle x, L^T L x\rangle + \langle L^T L x, L^T L x\rangle \\
&= \|x\|^2 - 2x^T L^T L x + x^T L^T L L^T L x \\
&= \|x\|^2 - x^T L^T L x \\
&= \|x\|^2 - \operatorname{tr}(x^T L^T L x) \\
&= \|x\|^2 - \operatorname{tr}(L x x^T L^T).
\end{aligned}$$

Thus, if we remove terms that do not depend on L, we can transform the problem in Eq. 2 into the following equivalent problem:

$$\max_{\substack{L \in \mathcal{M}_{d' \times d}(\mathbb{R}) \\ LL^T = I}} \quad \operatorname{tr}\left(L\Sigma L^T\right), \tag{3}$$

where $\Sigma = \sum_{i=1}^N x_i x_i^T$ is, except for a constant, the covariance matrix corresponding to the data in $\mathcal{X}$. This matrix is symmetric, and Theorem 11 guarantees that we can find a maximum of the problem if we build L adding the d' orthonormal eigenvectors corresponding to the d' largest eigenvalues of Σ. The directions that determine these vectors are the *principal directions*, and the components of the data transformed in the orthonormal system determined by the principal directions are the so-called *principal components*.

To conclude, the second approach from which the principal components problem can be dealt with consists of selecting the orthogonal directions for which the variance is maximized. We know that if Σ is the covariance matrix of $\mathcal{X}$, when applying a linear transformation L to the data, the new covariance matrix is given by $L\Sigma L^T$. If we want a transformation that reduces the dimensionality and for which the variance is maximized in each variable, what we are looking for is to take the trace of the previous matrix, which leads us back again to Eq. 3. The symmetry of Σ ensures that we can take the main orthonormal directions that maximize the variance for each possible value of d'.

Finally, it is important to note that the matrix $L \in \mathcal{M}_d(\mathbb{R})$ (taking all dimensions) that is constructed by adding Σ eigenvectors row by row is the orthogonal matrix that diagonalizes Σ, and therefore, when L is applied to the data, the transformed data have as the covariance matrix the diagonal matrix $L\Sigma L^T = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$, where $\lambda_1 \geq \dots \geq \lambda_d$ are the eigenvalues of Σ. This tells us that the eigenvalues of the covariance matrix represent the amount of variance explained by each of the principal directions. This provides an additional advantage to PCA, since it allows the percentage of variance that explains each principal component to be analyzed, in order to be able to later choose a dimension that adjusts to the amount of variance that we want to keep in the transformed data.

Figure 7 graphically exemplifies how principal component analysis works.

4.1.2 LDA

LDA (*linear discriminant analysis*) (Fisher, 1936) is a classical distance metric learning technique with the purpose of learning a projection matrix that maximizes the separation between classes in the projected space, that is, it tries to find the directions that best distinguish the different classes, as shown in Figure 8.

Figure 8 also allows us to compare the results of the projections obtained by PCA and LDA, showing the most remarkable difference between the two techniques: PCA does not take into account the labels information, while LDA does use it. We can observe that the directions obtained by PCA and LDA do not present any type of relationship, the latter being the only one of them that provides a data projection oriented to supervised learning.

It is also possible to observe in Figure 8 that it makes no sense to look for a second independent direction that continues to maximize class separation, while in PCA it always makes sense to look inductively for orthogonal directions that maximize variance. If the dataset shown in the figure had a third class, we could find a second direction that maximizes

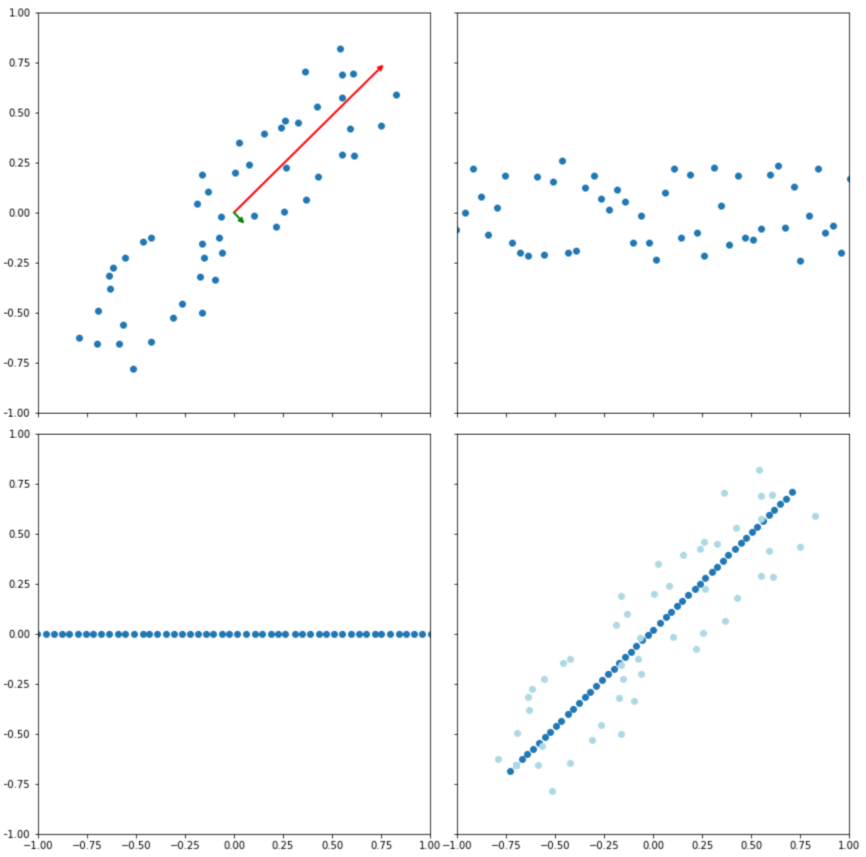

Figure 7: A graphical example of PCA. The first image shows a dataset, along with the principal directions (proportional according to the explained variance) learned by PCA. To the right, the data is projected at maximum dimension. We observe that this projection consists of rotating the data making the axes coincide with the principal directions. At the bottom left, data is projected onto the first principal component. Finally, to the right, the data recovered through the decompression matrix, along with the original data. We can see that the PCA projection is the one that minimizes the quadratic decompression error. In this particular case the decompressed data is on the regression line of the original data, due to the dimensions of the problem.

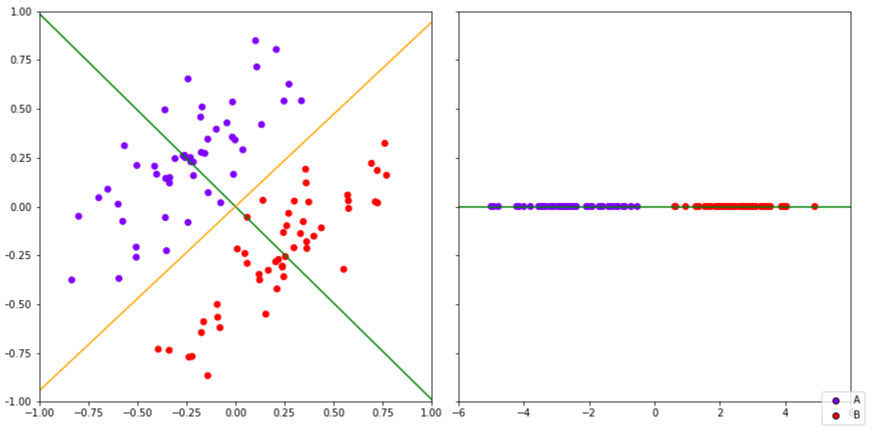

Figure 8: Graphical example of LDA and comparison with PCA. The first image shows a dataset, with the first principal direction determined by PCA, in orange, and the direction determined by LDA, in green. We observe that if we project the data on the direction obtained by LDA they separate, as it is shown in the right image. In contrast, the direction obtained by PCA only allows us to maximize the variance of the whole dataset, since it does not consider the information of the labels.

the separation between classes, thus offering the possibility of projecting onto a plane. In general, we will see that if we have r classes we will be able to find at most (and as long as the dimension of the original space allows it) $r-1$ directions that maximize the separation. This indicates that the projections that LDA is going to learn will be, in general, towards a quite low dimension, and always limited by the number of classes in the dataset.

Suppose we have the labeled dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, where $\mathcal{C}$ is the set of all the classes in the problem and $y_1, \dots, y_N \in \mathcal{C}$ are the corresponding labels. Suppose that the number of classes in the problem is $|\mathcal{C}| = r$. For each $c \in \mathcal{C}$ we define the set $\mathcal{C}_c = \{i \in \{1, \dots, N\} \colon y_i = c\}$, and $N_c = |\mathcal{C}_c|$. We consider the mean vector of each class,

$$\mu_c = \frac{1}{N_c} \sum_{i \in \mathcal{C}_c} x_i,$$

and the mean vector for the whole dataset,

$$\mu = \frac{1}{N} \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{C}_c} x_i = \frac{1}{N} \sum_{i=1}^{N} x_i.$$

We will define two scatter matrices, one *between-class*, denoted as S_b, and the other *within-class*, denoted as S_w. The between-class scatter matrix is defined as

$$S_b = \sum_{c \in \mathcal{C}} N_c (\mu_c - \mu)(\mu_c - \mu)^T.$$

And the within-class scatter matrix is defined as

$$S_w = \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{C}_c} (x_i - \mu_c)(x_i - \mu_c)^T.$$

Note that these matrices represent, except multiplicative constants, the covariances between the data of different classes, taking the class means as representatives for each class in the first case, and the sum, for each class, of the covariances of that class data, in the second case. Since we want to maximize the separation between classes we will formulate the problem of optimization as the search for a projection $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$ that maximizes the quotient of the between-class variances and within-class variances determined by the previous matrices. The problem is established as

$$\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad \operatorname{tr}\left((LS_wL^T)^{-1}(LS_bL^T)\right). \tag{4}$$

Theorem 12 assures us that, in order to maximize the problem given in Eq. 4, L has to be composed by the eigenvectors corresponding to the largest eigenvalues of $S_w^{-1}S_b$, as long as S_w is invertible. In practice, this happens in most problems where $N \gg d$, because S_w is the sum of N outer products, each of which may add a new dimension to the matrix rank. If $N \gg d$ it is likely that S_w is full-rank. This, together with the fact that S_w is positive semidefinite, would guarantee S_w to be positive definite, thus entering into the theorem hypothesis.

It is interesting to remark the similarity between the optimization problem in Eq. 4 and the expression of the Calinski-Harabasz index (Caliński and Harabasz, 1974), an index used in clustering to measure the separation of the established clusters, and that uses the same scatter matrices, and a similar quotient formulation.

Furthermore, let us note, as it was already mentioned at the beginning of this section, that at most we can get $r-1$ eigenvectors with a non zero corresponding eigenvalue. This is because the maximum rank of S_b is $r-1$, because its rank coincides with the rank of the matrix A that has as columns the vectors $\mu_c - \mu$ (we get $S_b = A\operatorname{diag}(N_{c_1}, \dots, N_{c_r})A^T$), which can have as maximum rank r, and this matrix also includes the linear combination $\sum N_c(\mu_c - \mu) = 0$, so at least one column is linearly dependent of the others. Therefore, $S_w^{-1}S_b$ also has a maximum rank of $r-1$. Consequently, the projection matrix that maximizes Eq. 4 is also going to have, at most, this rank, thus the projection will be contained in a space of this dimension. Therefore, the choice of a dimension $d' > r-1$ will not provide any additional information to that provided by the projection onto dimension $r-1$.

To conclude, although we have seen that LDA allows us to reduce dimensionality by adding supervised information as opposed to the non supervised PCA, it can also present some limitations:

- If the size of the dataset is too small, the within-class scatter matrix may be singular, preventing the calculation of $S_w^{-1}S_b$. In this situation, several mechanisms are proposed to keep this technique going. One of the most used consists of regularizing the problem, considering, instead of S_w, the matrix $S_w + \varepsilon I$, where $\varepsilon > 0$, making $S_w + \varepsilon I$ be positive definite. The problem of the singularity of S_w also arises if there are correlated attributes. This case can be avoided by eliminating redundant attributes in a preprocessing prior to learning.

- The definition of the scatter matrices assumes, to some extent, that the data in each class are distributed according to a multivariate gaussian distribution. Therefore, if the data presented other distributions, the projection learned might not be of enough quality.

- As already mentioned, LDA only allows the extraction of $r-1$ attributes, which may be suboptimal in some cases, as a lot of information could be lost.

4.1.3 ANMM

ANMM (*average neighborhood margin maximization*) (Wang and Zhang, 2007) is a distance metric learning technique specifically oriented to dimensionality reduction. It therefore follows the same path as the aforementioned PCA and LDA, trying to solve some of their limitations.

The objective of ANMM is to learn a linear transformation $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$, with $d' \leq d$, that projects the data onto a lower dimensional space, so that the similarity between the elements of the same class and the separation between classes is maximized, following the criterion of maximization of margins that we will show next.

We consider the training dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, with corresponding labels $y_1, \dots, y_N$, and we fix $\xi, \zeta \in \mathbb{N}$, and euclidean distance as the initial distance. From these variables we will create two types of neighborhoods.

Definition 27 *Let $x_i \in \mathcal{X}$.*

*We define the ξ-*nearest homogeneous neighborhood *of x_i as the set of the ξ samples in $\mathcal{X} \setminus \{x_i\}$ nearest to x_i that belong to its same class. We will denote it by $\mathcal{N}_i^o$.*

*We define the ζ-*nearest heterogeneous neighborhood *of x_i as the set of the ζ samples in $\mathcal{X}$ nearest to x_i that belong to a different class. We will denote it by $\mathcal{N}_i^e$.*

ANMM is intended to maximize the concept of *average neighborhood margin*, which we define below.

Definition 28 *Given $x_i \in \mathcal{X}$, its* average neighborhood margin *γ_i is defined as*

$$\gamma_i = \sum_{k\colon x_k \in \mathcal{N}_i^e} \frac{\|x_i - x_k\|^2}{|\mathcal{N}_i^e|} - \sum_{j\colon x_j \in \mathcal{N}_i^o} \frac{\|x_i - x_j\|^2}{|\mathcal{N}_i^o|}.$$

The (global) average neighborhood margin γ is defined as

$$\gamma = \sum_{i=1}^{N} \gamma_i.$$

Note that, for each $x_i \in \mathcal{X}$, its average neighborhood margin represents the difference between the average distance from x_i to its heterogeneus neighbors, and the average distance from x_i to its homogeneous neighbors. Therefore, maximizing this margin allows, locally, to move data from different classes away, and pulling those of the same class. Figure 9 graphically describes the concept of average neighborhood margin.

We are now looking for a linear transformation L that maximizes the margin associated with the projected data, $\{Lx_i \colon i = 1, \dots, N\}$. For such data, we have the average

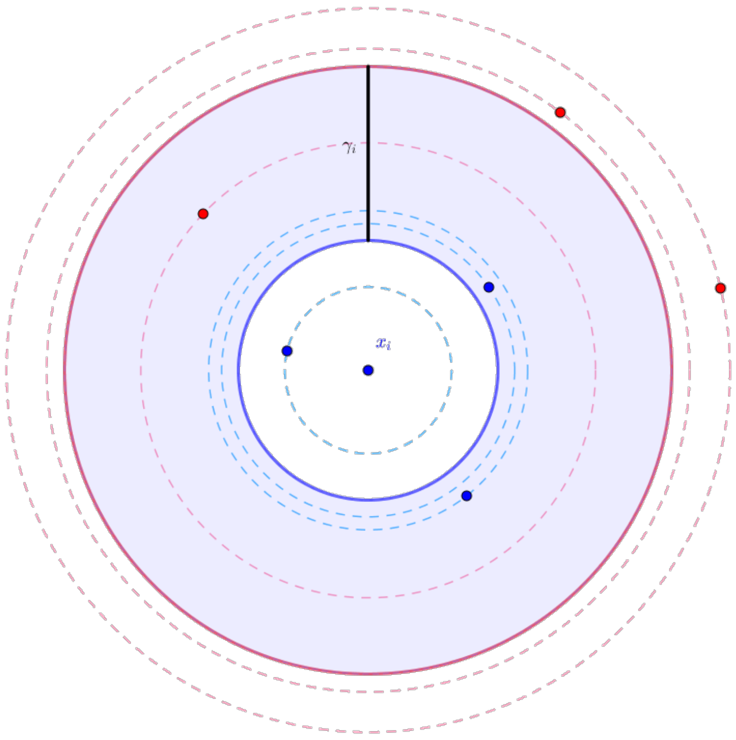

Figure 9: Graphical description of the average neighborhood margin, for the sample x_i, for $\xi = \zeta = 3$. The blue and red circumferences determine the average distance from x_i to data of the same and different classes, respectively.

neighborhood margin corresponding to that transformation,

$$\gamma^L = \sum_{i=1}^{N} \gamma_i^L = \sum_{i=1}^{N} \left(\sum_{k\colon x_k \in \mathcal{N}_i^e} \frac{\|Lx_i - Lx_k\|^2}{|\mathcal{N}_i^e|} - \sum_{j\colon x_j \in \mathcal{N}_i^o} \frac{\|Lx_i - Lx_j\|^2}{|\mathcal{N}_i^o|} \right).$$

Observe that, thanks to the linearity of the trace operator, we can express

$$\begin{aligned}
\sum_{i=1}^{n} \sum_{k\colon x_k \in \mathcal{N}_i^e} \frac{\|Lx_i - Lx_k\|^2}{|\mathcal{N}_i^e|} &= \operatorname{tr}\left(\sum_{i=1}^{N} \sum_{k\colon x_k \in \mathcal{N}_i^e} \frac{(Lx_i - Lx_k)(Lx_i - Lx_k)^T}{|N_i^e|} \right) \\
&= \operatorname{tr}\left[L \left(\sum_{i=1}^{N} \sum_{k\colon x_k \in \mathcal{N}_i^e} \frac{(x_i - x_k)(x_i - x_k)^T}{|N_i^e|} \right) L^T \right] \\
&= \operatorname{tr}(LSL^T),
\end{aligned}$$

where $S = \sum_i \sum_{k\colon x_k \in \mathcal{N}_i^e} \frac{(x_i - x_k)(x_i - x_k)^T}{|\mathcal{N}_i^e|}$ is called the *scatter matrix*. In a similar way, if we define $C = \sum_i \sum_{j\colon x_j \in \mathcal{N}_i^o} \frac{(x_i - x_j)(x_i - x_j)^T}{|\mathcal{N}_i^o|}$, which we will call the *compactness matrix*, we get

$$\sum_{i=1}^{n} \sum_{j\colon x_j \in \mathcal{N}_i^o} \frac{\|Lx_i - Lx_j\|^2}{|\mathcal{N}_i^o|} = \operatorname{tr}(LCL^T).$$

And therefore, combining both expressions,

$$\gamma^L = \operatorname{tr}(L(S - C)L^T). \tag{5}$$

The maximization of γ^L as presented in Eq. 5 is not restrictive enough, because it is enough to multiply L by positive constants to get a value of γ^L as large as we want. That is why the constraint $LL^T = I$ is added, so we end up with the next optimization problem:

$$\begin{aligned}
\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad & \operatorname{tr}\left(L(S - C)L^T\right) \\
\text{s.t.:} \quad & LL^T = I.
\end{aligned}$$

Observe that $S - C$ is symmetric, as it is the difference between two positive semidefinite matrices (each of them is the sum of outer products). Theorem 11 tells us that the matrix L we are looking for can be built by adding, by rows, the d' eigenvectors of $S - C$ corresponding to its d' largest eigenvalues.

To conclude, note that ANMM solves some of the issues of the previously mentioned PCA and LDA. On the one hand, it is a supervised learning algorithm, hence it uses the class information that is ignored by PCA. On the other hand, faced with the shortcomings of LDA, we can see that:

- It does not have computational problems with small samples, for which scatter or compactness matrices may be singular, because it does not have to calculate their inverse matrices.

- It does not make any assumption about the class distributions. The formulation of the problem is purely geometric.

- It admits any size for dimensionality reduction. It does not impose that this size must be lower than the number of classes.

Finally, we can also observe that, if we keep the maximum dimension d, the condition $LL^T = I$ implies that L is orthogonal and $L^T L = I$, thus we are just learning an isometry, as already happened with PCA. Therefore, distance-based classifiers will only be able to experience improvements when the chosen dimension is strictly smaller than the original one.

4.2 Algorithms to Improve Nearest Neighbors Classifiers

In the following paragraphs we will analyze algorithms specifically designed to work with nearest neighbors classifiers. The algorithms we will study are known as LMNN (Weinberger and Saul, 2009) and NCA (Goldberger et al., 2005).

4.2.1 LMNN

LMNN (*large margin nearest neighbors*) (Weinberger and Saul, 2009) is a distance metric learning algorithm aimed specifically at improving the accuracy of the k-nearest neighbors classifier. It is based on the premise that this classifier will label a sample more reliably if its k neighbors share the same label, and to do so it tries to learn a distance that maximizes the number of samples that share its label with as many neighbors as possible.

In this way, the LMNN algorithm tries to minimize an error function that penalizes, on the one hand, the large distance between each sample and those considered its ideal neighbors, and on the other hand, the small distances between examples of different classes.

Suppose we have a dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ with corresponding labels $y_1, \dots, y_N$. To work, the algorithm makes use of the concept of *target neighbors*. Given a sample $x_i \in \mathcal{X}$, its k target neighbors are those examples of the same class as x_i and different from this, for which it is desired to be considered as neighbors in the nearest neighbors classification. If x_j is a target neighbor of x_i, then we will write it as $j \rightsquigarrow i$. Observe that the relationship given by $\rightsquigarrow$ may not be symmetric. Target neighbors are fixed during the learning process. If we have some prior information about our dataset we can use it to determine the target neighbors. Otherwise, a good option is to use the nearest neighbors for the euclidean distance as target neighbors.

Once the target neighbors have been established, for each distance and for each sample in $\mathcal{X}$ we can create a perimeter determined by the the furthest target neighbor. We are looking for distances for which there are no samples of other classes in this perimeter. It is necessary to emphasize that with this perimeter there are not enough separation guarantees, because a feasible distance could have collapsed all the target neighbors in a point, and then the perimeter would have radius zero. For this reason, a margin determined by the radius of the perimeter is considered, to which a positive constant is added. We will see that there is no loss of generality, because of the function that we will define, in supposing that this constant is 1. Any sample of a different class that invades this margin will be called an *impostor*. Our objective, therefore, will be, in addition to bringing each sample as close as possible to its target neighbors, to try to keep impostors as far away as possible.

In mathematical terms, if our distance is determined by the linear transformation $L \in \mathcal{M}_d(\mathbb{R})$, and $x_i, x_j \in \mathcal{X}$ with $j \rightsquigarrow i$, we will say that x_l is an impostor for these samples if $y_l \neq y_i$ and $\|L(x_i - x_j)\|^2 \leq \|L(x_i - x_j)\|^2 + 1$. In Figure 10 the concepts of target neighbor and impostor are graphically described. Finally, note that the margin is defined in terms of the squared distances, instead of considering only the distance. This will make the problem formulation easy to solve.

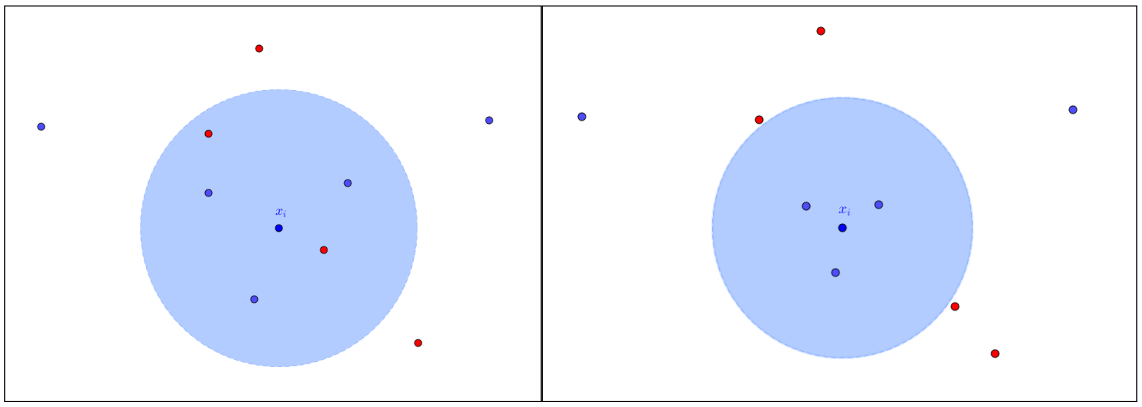

Figure 10: Graphical description of target neighbors and impostors (with $k = 3$) for the sample x_i. The blue circle represents the margin determined by the target neighbors. All the points of different classes in this circle are impostors. LMNN's goal will be to bring the target neighbors as close as possible and to remove the impostors from the circle. Therefore, data of the same class that are not target neighbors will not have any influence, and impostors will no longer be penalized as soon as they leave the margin, as shown in right image. This gives a local nature to this learning technique.

We now proceed to define accurately the terms of the objective function. As already mentioned, it will be composed of two terms. The first one will penalize distant target neighbors and the second one will penalize nearby impostors. The first term is defined as

$$\varepsilon_{pull}(L) = \sum_{i=1}^{N} \sum_{j \rightsquigarrow i} \|L(x_i - x_j)\|^2.$$

The minimization of this error causes a pulling force between the data samples. The second term is defined as

$$\varepsilon_{push}(L) = \sum_{i=1}^{N} \sum_{j \rightsquigarrow i} \sum_{l=1}^{N} (1 - y_{il})[1 + \|L(x_i - x_j)\|^2 - \|L(x_i - x_l)\|^2]_+,$$

where y_{il} is a binary variable which takes the value 1 if $y_i = y_l$, and 0 if $y_i \neq y_l$, and the operator $[\cdot]_+ \colon \mathbb{R} \to \mathbb{R}_0^+$ is defined as $[z]_+ = \max\{z, 0\}$. Thus, this error adds up when $y_{il} = 0$ (that is, x_l is in different class to x_i), and the second factor is strictly positive (that is, the margin defined by the target neighbors is exceeded). The minimization of this second term causes a pushing force between the data samples.

Finally, the objective function results from combining these two terms. After fixing $\mu \in]0,1[$, we define

$$\varepsilon(L) = (1-\mu)\varepsilon_{pull}(L) + \mu\varepsilon_{push}(L). \tag{6}$$

The authors state that, experimentally, the choice of μ does not cause great differences in results, so it is usually taken $\mu = 1/2$. Minimizing this function will lead us to learn the distance we were looking for. Note that this function is sub-differentiable, but not convex, so if we use a subgradient descent method under this approach we may be stuck in a local optimal. However, we can reformulate the objective function in order to make it act over the positive semidefinite cone. If for every $L \in \mathcal{M}_d(\mathbb{R})$ we take $M = L^TL \in S_d(\mathbb{R})_0^+$, we know that $\|x_i - x_j\|_M^2 = \|L(x_i - x_j)\|_2^2$, and consequently,

$$\varepsilon(M) = (1-\mu)\sum_{i=1}^{N}\sum_{j\rightsquigarrow i}\|x_i - x_j\|_M^2 + \mu\sum_{i=1}^{N}\sum_{j\rightsquigarrow i}\sum_{l=1}^{N}[1 + \|x_i - x_j\|_M^2 - \|x_i - x_l\|_M^2]_+ \tag{7}$$

is a convex function in M that takes the same values as $\varepsilon(L)$. The minimization of $\varepsilon(M)$ in this case is subject to the constraint $M \in S_d(\mathbb{R})_0^+$, so the projected subgradient method, with projections onto the positive semidefinite cone, can be used to optimize this function. In addition, we can easily calculate a subgradient $G \in \partial\varepsilon/\partial M$ given by

$$G = (1-\mu)\sum_{i,j\rightsquigarrow i} O_{ij} + \mu \sum_{(i,j,l)\in\mathcal{N}} (O_{ij} - O_{il}),$$

where $\mathcal{N}$ is the set of triplets (i,j,l) for which x_l is an impostor over x_i with the margin determined by x_j, and $O_{ij} = (x_i - x_j)(x_i - x_j)^T$ are the outer products obtained from the distances differentiation. The first term of the gradient is constant, while the second term only varies in each iteration with the changes of the impostors that enter or leave the set $\mathcal{N}$. These considerations allow a fairly efficient gradient calculation.

As for dimensionality reduction, two different alternatives are presented. If we keep the optimization with respect to M, it is not feasible to add rank restrictions, as we saw in Example 2. Therefore, the use of PCA is suggested prior to the algorithm execution, to project the data onto its first principal components, and then apply LMNN on the projected data. The other alternative is to optimize the objective function with respect to $L \in \mathcal{M}_{d'\times d}(\mathbb{R})$, with $d' < d$ using a gradient descent algorithm. In this case the optimization is not convex, but we learn directly a linear transformation that reduces the dimensionality without making changes in the optimization of Eq. 6. Authors also state, based on empirical results, that this non-convex optimization gives good results.

Other proposals made for the improvement of this algorithm consist of applying LMNN multiple times, learning new metrics each time, and using these metrics to determine increasingly accurate target neighbors, or learning different metrics locally. Finally, although the distance learned by LMNN is designed to be used by the k-neighbors classifier, it is possible to use the objective function itself as a classification method. These classification models are called *energy-based*. Thus, to classify a test sample x_t, for each possible label value y_t, we look for k target neighbors in the training set for class y_t, and evaluate the *energy* for the metric learned, finally assigning to x_t the value of y_t that provides the lowest energy. According to the objective function, energy will penalize large distances between x_t and its target neighbors, impostors on the x_t perimeter, and perimeters of other classes invaded by x_t. Therefore,

$$\begin{aligned} y_t^{pred} = \arg\min_{y_t} \Bigg\{ & (1-\mu)\sum_{j\rightsquigarrow t} \|x_t - x_j\|_M^2 \\ & + \mu \sum_{j\rightsquigarrow t,l} (1-y_{tl}) \left[1 + \|x_t - x_j\|_M^2 - \|x_t - x_l\|_M^2\right]_+ \\ & + \mu \sum_{i,j\rightsquigarrow i} (1-y_{it}) \left[1 + \|x_i - x_j\|_M^2 - \|x_i - x_t\|_M^2\right]_+ \Bigg\}. \end{aligned}$$

4.2.2 NCA

NCA (*neighborhood component analysis*) (Goldberger et al., 2005) is another distance metric learning algorithm aimed specifically at improving the accuracy of the nearest neighbors classifiers. Its aim is to learn a linear transformation with the goal of minimizing the leave-one-out error expected by the nearest neighbor classification. Additionally, this transformation could be used to reduce the dimensionality of the dataset, and thus make the classifier more efficient.

We consider the training set $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, labeled by $y_1, \dots, y_N$. We want to learn a distance, determined by a linear transformation $L \in \mathcal{M}_d(\mathbb{R})$, that optimizes the accuracy of the nearest neighbors classifier. Ideally, we would optimize the performance of the classifier over the test dataset, but we only have the training set. Therefore, our goal will be to try to optimize the classification leave-one-out error on the training set. The choice of the leave-one-out error is due to the nature of the nearest neighbors classifier: as we will learn and evaluate over the same set, the nearest neighbor of each sample would be the sample itself, which would not allow the results to be interpreted correctly if the sample is kept while evaluating it.

However, the function that maps each transformation L to the leave-one-out error for the distance corresponding to L has no guarantee of differentiability, not even continuity, so it is not easy to deal with it for optimization (observe that the image of this function is a finite set, and its domain is a connected set, so it cannot be continous unless it is constant, which does not happen in non-trivial examples).

To do this, NCA tries to approach the problem in a stochastic way, that is, instead of operating with the leave-one-out error directly, it operates with its expected value for the probability that we will define below.

Given two samples $x_i, x_j \in \mathcal{X}$, we define the probability that x_i has x_j as its nearest neighbor, for the distance determined by the mapping L, as follows:

$$p_{ij}^L = \frac{\exp\left(-\|Lx_i - Lx_j\|^2\right)}{\sum\limits_{k\neq i} \exp\left(-\|Lx_i - Lx_k\|^2\right)} \quad (j \neq i), \qquad p_{ii}^L = 0.$$

Notice that, indeed, p_{i*} defines a probability measure on the set $\{1,\dots,N\}$, for each $i \in \{1,\dots,N\}$. Under this probability law, we can define the probability that the sample x_i is correctly classified as the sum of the probabilities that x_i has as its nearest neighbor each sample of its same class, that is

$$p_i^L = \sum_{j\in C_i} p_{ij}^L, \text{ where } C_i = \{j \in \{1,\dots,N\} \colon y_j = y_i\}.$$

Finally, the expected number of correctly classified samples, and the function we will try to maximize, is obtained as

$$f(L) = \sum_{i=1}^N p_i^L = \sum_{i=1}^N \sum_{j\in C_i} p_{ij}^L = \sum_{i=1}^N \sum_{\substack{j\in C_i\\ j\neq i}} \frac{\exp\left(-\|Lx_i - Lx_j\|^2\right)}{\sum\limits_{k\neq i}\exp\left(-\|Lx_i - Lx_k\|^2\right)}.$$

This function is differentiable, and its derivative can be computed as

$$\nabla f(L) = 2L\sum_{i=1}^N \left(p_i^L \sum_{k=1}^N p_{ik}^L O_{ik} - \sum_{j\in C_i} p_{ij}^L O_{ij} \right),$$

where $O_{ij} = (x_i - x_j)(x_i - x_j)^T$ represent again the outer products between the differences of the samples in $\mathcal{X}$. Once the gradient is known, we can optimize the objective function using a gradient ascent method. Note that the objective function is not concave, and can therefore be trapped in local optima. Another issue for this algorithm is the possibility of overfitting, if the expected leave-one-out error of the learned distance is too low. Authors affirm, based on the experimentaal results, that normally there is no overfitting, even if we ascend a lot in the objective function.

4.3 Algorithms to Improve Nearest Centroids Classifiers

In this block we will analyze, following the previous lines, algorithms specifically oriented to improve distance-based classifiers, focusing in this case on the classifiers based on centroids. The algorithms we will study are NCMML and NCMC (Mensink et al., 2012).

4.3.1 NCMML

NCMML (*nearest class mean metric learning*) (Mensink et al., 2012) is a distance metric learning algorithm specifically designed to improve the nearest class mean (NCM) classifier. To do this, it uses a probabilistic approach similar to that used by NCA to improve the accuracy of the nearest neighbors classifier.

Nearest class mean classifier, during learning process, calculates the mean vectors of each class subset. Then, when predicting a new sample, it assigns the class of the nearest mean vector found. It is a very efficient and simple classifier, although its simplicity makes it a rather weak classifier against datasets that are not grouped around their mean. We will learn in the following lines how to learn a distance for this classifier.

We consider the training set $\mathcal{X} = \{x_1,\dots,x_N\} \subset \mathbb{R}^d$, with labels $y_1,\dots,y_N \in \mathcal{C}$, where $\mathcal{C} = \{c_1,\dots,c_r\}$ is the set of available classes. For each $c \in \mathcal{C}$, we call $\mu_c \in \mathbb{R}^d$ the mean vector of the samples belonging to the class c, that is, $\mu_c = \frac{1}{N_c}\sum_{i\colon y_i=c} x_i$, where N_c is the number of elements of $\mathcal{X}$ that belong to class c. Given a linear transformation $L \in \mathcal{M}_{d'\times d}(\mathbb{R})$, we will define, for each $x \in \mathcal{X}$ and each $c \in \mathcal{C}$, the probability that x will be labeled with the class c (according to the nearest class mean criterion) as follows:

$$p_L(c|x) = \frac{\exp\left(-\frac{1}{2}\|L(x-\mu_c)\|^2\right)}{\sum\limits_{c'\in\mathcal{C}} \exp\left(-\frac{1}{2}\|L(x-\mu_{c'})\|^2\right)}.$$

Note that $p_L(\cdot|x)$ effectively defines a probability in the set $\mathcal{C}$. Once the above probability is defined, the objective function that NCMML tries to maximize is the log-likelihood for the labeled data in the training set, that is,

$$\mathcal{L}(L) = \frac{1}{N}\sum_{i=1}^{N} \log p_L(y_i|x_i).$$

This function is differentiable and its gradient is given by

$$\nabla\mathcal{L}(L) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c\in\mathcal{C}} \alpha_{ic} L(\mu_c - x_i)(\mu_c - x_i)^T,$$

where $\alpha_{ic} = p_L(c|x_i) - [\![y_i = c]\!]$ and $[\![R]\!]$ denotes the indicator function for the condition R. The maximization of this function using gradient methods is the task carried out by NCMML.

4.3.2 NCMC

Although nearest class mean classifier is a simple, intuitive and efficient classifier in both learning and prediction processes, it has one major drawback, and that is that it assumes that classes are grouped around their center, which is an overly restrictive hypothesis. In Figure 11 we can see an example where NCM is unable to give good results.

One way to solve this problem is, instead of considering the center of each class to classify new samples, to find subgroups within each class that present a quality grouping, and to consider the center for each of its subgroups. In this way we would have a set of centroids for each class, and at the time of classifying a new sample, it would suffice to select the nearest centroid and assign it the class of which it is centroid.

In this new classifier, which we will call NCMC (*nearest class with multiple centroids*), the clustering algorithms come into play. There are numerous algorithms (Xu and Wunsch, 2005) to obtain a set of clusters from a dataset, each with its advantages and disadvantages. Due to the form of our problem, in which we are interested not only in obtaining a set of clusters for each class, but also a center for each cluster, the algorithm that meets the most suitable conditions, besides being simple and efficient is k-Means.

To classify with NCMC, the use k-Means reduces to applying the segmentation algorithm within each subset of data associated with each of the classes of the problem. In this way, we obtain in a simple way the set of centroides we wanted for each class, and on which we can carry out the classification of new data simply by searching for the nearest centroid.

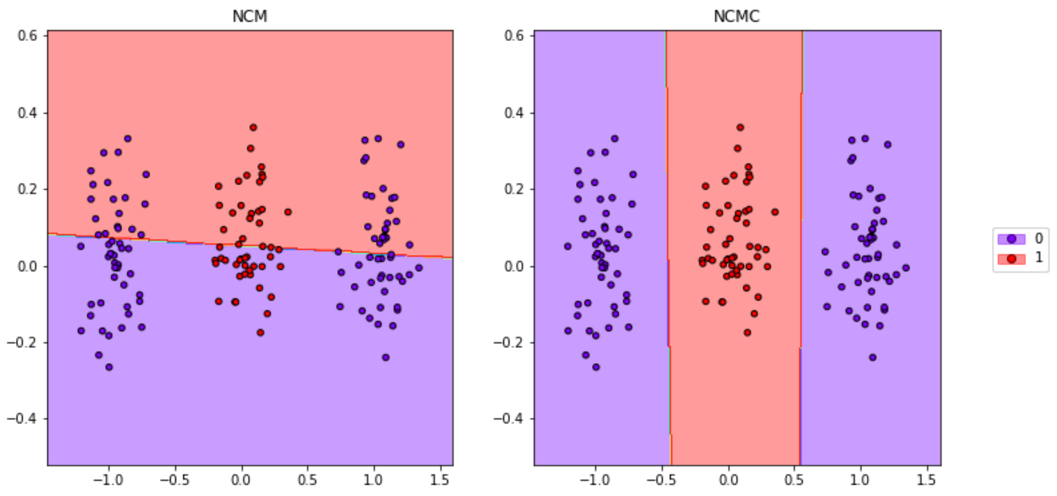

Figure 11: Dataset where the NCM classifier does not provide good results, because the centroids of both classes are very close and both fall between the points of class 1. We will see that, by choosing more than one centroid in an appropiate way, we can classify this set as shown in right image.

For this algorithm, as it happens with k-Means, it is necessary to previously establish the number of centroids for each class. These numbers can be estimated by cross validation.

Once the NCMC classifier is defined, the distance learning process (Mensink et al., 2012) is similar to NCM. Following the notation used in NCMML, in this case, instead of a set of class centers $\{\mu_c\}$, with $c \in \mathcal{C}$, we have a set of centroids, $\{m_{c_j}\}_{j=1}^{k_c}$, with $k_c \in \mathbb{N}$, for each $c \in \mathcal{C}$. In this case, the probabilities associated with each class for the correct prediction of $x \in \mathcal{X}$ are given by $p_L(c|x) = \sum_{j=1}^{k_c} p_L(m_{c_j}|x)$, where the centroids are those whose probability is defined by the softmax function

$$p_L(m_{c_j}|x) = \frac{\exp\left(-\frac{1}{2}\|L(x - m_{c_j})\|^2\right)}{\sum\limits_{c \in \mathcal{C}} \sum\limits_{i=1}^{k_c} \exp\left(-\frac{1}{2}\|L(x - m_{c_i})\|^2\right)}.$$

Again, we maximize the log-likelihood function $\mathcal{L}(L) = \frac{1}{N}\sum_{i=1}^{N} p_L(y_i|x_i)$, whose gradient is given by

$$\nabla\mathcal{L}(L) = \frac{1}{N}\sum_{i=1}^{N}\sum_{c \in \mathcal{C}}\sum_{j=1}^{k_c} \alpha_{ic_j} L(m_{c_j} - x_i)(m_{c_j} - x_i)^T,$$

where

$$\alpha_{ic_j} = p_L(m_{c_j}|x_i) - [\![y_i = c]\!] \frac{p_L(m_{c_j}|x_i)}{\sum_{j'=1}^{k_c} p_L(m_{c_{j'}}|x_i)}.$$

The log-likelihood maximization by gradient methods is the task carried out by the distance learning technique for NCMC classifier, which we will call with the same name as the classifier.

4.4 Information Theory Based Algorithms

In this section we will study several distance metric learning algorithms based on information theory, specifically, in the Kullback-Leibler and Jeffrey divergences. Their working scheme is similar. First of all, they establish different probability distributions on the data, and then they try to bring them closer or further away by using the divergences. The algorithms we will study are ITML (Davis et al., 2007), DMLMJ (Nguyen et al., 2017) and MCML (Globerson and Roweis, 2006).

4.4.1 ITML

ITML (*information theoretic metric learning*) (Davis et al., 2007) is a distance metric learning technique whose objective is to find a metric as close as possible to an initial distance, understanding this closeness from the point of view of relative entropy, as we will formulate later, making that metric satisfy certain similarity constraints for the trained data.

ITML starts with a dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, not necessarily labeled, but for which it is known that certain pairs of samples considered similar must be at a distance lower than or equal to u, and other pairs of samples considered not similar must be at a distance greater than or equal to l, where $u, l \in \mathbb{R}^+$ are pre-defined constants, with relative small and large values, respectively, with respect to the dataset.

From the data with the indicated restrictions, ITML considers an initial distance corresponding to a positive definite matrix M_0, and tries to find a positive definite matrix M, as similar as possible to M_0, and that respects the imposed similarity constraints. The way to measure the similarity between M and M_0 is done using information theory tools.

As we saw in Section 2.3, there is a correspondence between positive definite matrices and multivariate gaussian distributions, if we fix the same mean vector μ for every distribution. Given $M \in S_d(\mathbb{R})^+$ we can then construct a normal distribution through its density function,

$$p(x|M) = \frac{1}{(2\pi)^{n/2}\det(M)^{1/2}} \exp\left((x-\mu)^T M^{-1}(x-\mu)\right).$$

Reciprocally, from this distribution, if we calculate the covariance matrix, we recover the matrix M. Using this correspondence, we will measure the closeness between M_0 and M through the Kullback-Leibler divergence between their corresponding gaussian distributions, that is,

$$\text{KL}(p(x|M_0))\|p(x|M)) = \int p(x|M_0) \log \frac{p(x|M_0)}{p(x|M)} dx.$$

Once we have defined the mechanism to measure the proximity between the metrics, we can formulate the optimization problem of the technique ITML. If we call S and D to the sets of pairs of indices on the elements of $\mathcal{X}$ that represent the samples considered similar and not similar, respectively, and we start from the initial metric M_0, the problem is

$$\begin{aligned} \min_{M \in S_d(\mathbb{R})^+} \quad & \mathrm{KL}(p(x|M_0) \| p(x|M)) \\ \text{s.t.:} \quad & d_M(x_i, x_j) \leq u, \quad (i,j) \in S \\ & d_M(x_i, x_j) \geq l, \quad (i,j) \in D. \end{aligned} \tag{8}$$

We have seen in Theorem 20 that the Kullback-Leibler divergence between two gaussian distributions with the same mean can be expressed in terms of the *log-det* matrix divergence. This allows us to reformulate Eq. 8 in a way that is easier to deal with computationally:

$$\begin{aligned} \min_{M \in S_d(\mathbb{R})^+} \quad & D_{ld}(M_0 \| M) \\ \text{s.t.:} \quad & \mathrm{tr}(M(x_i - x_j)(x_i - x_j)^T) \leq u, \quad (i,j) \in S \\ & \mathrm{tr}(M(x_i - x_j)(x_i - x_j)^T) \geq l, \quad (i,j) \in D. \end{aligned} \tag{9}$$

We may not be able to find a metric M that simultaneously satisfies every constraint, so the problem may not have a solution. Therefore, ITML introduces in Eq. 9 slack variables through which we obtain a problem whose optimization establishes a trade-off between the minimization of the divergence and the fulfillment of the constraints, in order to arrive to an approximate solution of the original problem, in case there is no solution for this. Finally, the computational technique used in the resolution of this optimization problem is the *Bregman projections method* discussed in Section 2.1.2.

4.4.2 DMLMJ

DMLMJ (*distance metric learning through the maximization of the Jeffrey divergence*) (Nguyen et al., 2017) is another distance metric learning technique based on information theory. In this case, the tool that is used by DMLMJ is the Jeffrey divergence, to separate as much as possible the distribution associated to similar points from that associated to dissimilar points, in the sense that we will see below.

We consider the training set $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$ with corresponding labels $y_1, \dots, y_N$, and we set $k \in \mathbb{N}$. As we have already commented, DMLMJ tries to maximize, with respect to the Jeffrey divergence, the separation between distributions of similar and not similar points. To do this, we will introduce several concepts.

Definition 29 *Given $x_i \in \mathcal{X}$, the k-*positive neighborhood *of x_i is defined as the set of the k nearest neighbors of x_i in $\mathcal{X} \setminus \{x_i\}$ whose class is the same as x_i. It is denoted by $V_k^+(x_i)$.*

The k-negative neighborhood of x_i is defined as the set of the k nearest neighbors of x_i in $\mathcal{X}$ whose class is different from that of x_i. It is denoted by $V_k^-(x_i)$.

*The k-*positive difference space *of the labeled dataset is defined as the set*

$$S = \{x_i - x_j \colon x_i \in \mathcal{X}, x_j \in V_k^+(x_i)\}.$$

*Similarly, the k-*negative difference space *of the labeled dataset is defined as the set*

$$D = \{x_i - x_j \colon x_i \in \mathcal{X}, x_j \in V_k^-(x_i)\}.$$

Sets S and D represent, therefore, the vectors with the differences between the samples in $\mathcal{X}$ and its k nearest neighbors, from the same or a different class, respectively. We refer to P and Q as the probability distributions in the spaces S and D, respectively, assuming that they are multivariate gaussians. We will also assume that both distributions have zero mean. This assumption is reasonable, since in practice, in most cases, if x_i is a neighbor of x_j, x_j is also a neighbor of x_i, then both differences will appear in the difference space, averaging zero. Finally, we will call the corresponding covariance matrices Σ_S and Σ_D, respectively.

If we now apply a linear transformation to the data, $x \mapsto Lx$, with $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$, the transformed distributions will still have mean zero, and covariances $L\Sigma_S L^T$ and $L\Sigma_D L^T$, respectively. We will call these distributions P_L and Q_L. The goal of DMLMJ is to find a transformation that maximizes the Jeffrey divergence between P_L and Q_L, that is, the problem to optimize is:

$$\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad f(L) = \mathrm{JF}(P_L \| Q_L) = \mathrm{KL}(P_L \| Q_L) + \mathrm{KL}(Q_L \| P_L).$$

As it was shown in Proposition 22, Jeffrey divergence between the gaussian distributions P_L and Q_L can be rewritten as

$$f(L) = \frac{1}{2} \operatorname{tr}\left((L\Sigma_S L^T)^{-1}(L\Sigma_D L^T) + (L\Sigma_D L^T)^{-1}(L\Sigma_S L^T)\right) - d'.$$

Since d' is constant, we obtain the equivalent problem

$$\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad J(L) = \operatorname{tr}\left((L\Sigma_S L^T)^{-1}(L\Sigma_D L^T) + (L\Sigma_D L^T)^{-1}(L\Sigma_S L^T)\right).$$

Theorem 13 tells us that, to maximize $J(L)$, we can choose the d' eigenvectors of $\Sigma_S^{-1}\Sigma_D$, $v_1, \dots, v_{d'}$ corresponding to the largest values of $\lambda_i + 1/\lambda_i$, with λ_i being the eigenvalue of $\Sigma_S^{-1}\Sigma_D$ associated with v_i, and add this eigenvectors to the rows of L. The transformation L constructed from these eigenvectors determines the distance that is learned by the DMLMJ technique.

Finally, the only additional requirement necessary to complete the construction of L is the calculation of the covariance matrices Σ_S and Σ_D. Bearing in mind that it has been assumed that the mean of the distributions of S and D is 0, we can obtain these matrices quite simply from the difference vectors, as shown below:

$$\Sigma_S = \frac{1}{|S|} \sum_{i=1}^{N} \left[\sum_{x_j \in V_k^+(x_i)} (x_i - x_j)(x_i - x_j)^T \right],$$

$$\Sigma_D = \frac{1}{|D|} \sum_{i=1}^{N} \left[\sum_{x_j \in V_k^-(x_i)} (x_i - x_j)(x_i - x_j)^T \right].$$

Let us observe that we can also see this algorithm as a dimensionality reduction algorithm and even as an algorithm oriented to improve the nearest neighbors classifier, due to its local character.

4.4.3 MCML

MCML (*maximally collapsing metric learning*) (Globerson and Roweis, 2006) is a supervised distance metric learning technique, based on the idea that if all the samples of the same class were projected to the same point, and data of different classes were projected to different points and sufficiently far away, we would have, over the projected data, an ideal class separation. Its purpose is to learn a distance metric that allows to collapse as much possible, within the limitations of the metric, all the samples of the same class in a single point, arbitrarily far from the points where the samples of the remaining classes will collapse.

We consider the dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, with corresponding labels $y_1, \dots, y_N$. We want to learn a metric determined by $M \in S_d(\mathbb{R})^+$ that tries to collapse the classes as much as possible according to the approach of the previous paragraph. The way to deal with this problem will consist once again in using the tools provided by the information theory. To do this, we first introduce a conditional distribution on the points of the dataset, analogous to that established in the case of NCA. If $i, j \in \{1, \dots, N\}$, with $i \neq j$, we define the probability that x_j will be classified with the class of x_i according to the distance between x_i and x_j as follows:

$$p^M(j|i) = \frac{\exp(-\|x_i - x_j\|_M^2)}{\sum\limits_{k \neq i} \exp(-\|x_i - x_k\|_M^2)}.$$

Furthermore, the ideal distribution we are looking for is a binary distribution for which the probability that a sample is correctly classified is 1, and 0 otherwise, that is,

$$p_0(j|i) \propto \begin{cases} 1, & y_i = y_j \\ 0, & y_i \neq y_j \end{cases}.$$

Note that during the training process we know the real classes of the data, therefore we can deal with this last probability. Besides, we can observe that if we get a metric M whose associated distribution p^M coincides with p_0, then, under very mild sufficiency conditions on the data, we will be able to collapse the classes in infinitely distant points.

Indeed, suppose there are at least $r + 2$ samples in each class, were r is the rank of M, and that $p^M(j|i) = p^0(j|i)$ for any $i, j \in \{1, \dots, N\}$. Then, on the one hand, from $p^M(j|i) = 0$ for $y_i \neq y_j$, it follows that $\exp(-\|x_i - x_j\|_M^2) = 0$, which undoubtedly leads to x_i and x_j being infinitely distant when their classes are different. On the other hand, from $p^M(i|j) \propto 1$ for any x_i, x_j with $y_i = y_j$, it follows that the value $\exp(-\|x_i - x_j\|_M^2)$ is constant for all the members of the same class, and consequently, all the points in the same class are equidistant. As M has rank r, it is inducing a distance on a subspace of dimension r, where it is known that at most there can be $r+1$ different points and equidistant between them. Since we are assuming that there are at least $r + 2$ points per class, all the points of the same class must have a distance of 0 between them with respect to M, thus collapsing into a single point.

Once both distributions are set, the objective of MCML is, as we have already commented, to approximate $p^M(\cdot|i)$ to $p_0(\cdot|i)$ as much as possible, for each i, using the relative entropy between both distributions. The optimization problem is, therefore, to minimize this divergence,

$$\min_{M \in S_d(\mathbb{R})_0^+} \quad f(M) = \sum_{i=1}^{N} \mathrm{KL}\left[p_0(\cdot|i) \| p^M(\cdot|i)\right].$$

We can rewrite the objective function in terms of elementary functions:

$$\begin{aligned} f(M) &= \sum_{i=1}^{N}\sum_{j=1}^{N} p_0(j|i)\log\frac{p_0(j|i)}{p^M(j|i)} = \sum_{i=1}^{N}\sum_{j\colon y_i=y_j}\log\frac{1}{p^M(j|i)} \\ &= \sum_{i=1}^{N}\sum_{j\colon y_i=y_j} -\log p^M(j|i) \\ &= -\sum_{i=1}^{N}\sum_{j\colon y_i=y_j}\left(-\|x_i-x_j\|_M^2 - \log\sum_{k\neq i}\exp(-\|x_i-x_k\|^2)\right) \\ &= \sum_{i=1}^{N}\sum_{j\colon y_i=y_j}\|x_i-x_j\|_M^2 + \sum_{i=1}^{N}\log\sum_{k\neq i}\exp(-\|x_i-x_k\|^2). \end{aligned} \tag{10}$$

This function is differentiable, and each summand of the previous expression is convex in M, the first because it is a distance function in M (which is affine), and the second because it is a *log-sum-exp* function (see Boyd and Vandenberghe, 2004, sec. 3.1.5) composed with a distance function. In addition, the restriction $M \in S_d(\mathbb{R})_0^+$ is convex, so we can use the projected gradient descent algorithm with projections onto the positive semidefinite cone to optimize the objective function. This requires an expression of the gradient of the objective function, which can be calculated from its expression in Eq. 10:

$$\nabla f(M) = \sum_{i,j\colon y_i=y_j}(x_i-x_j)^T(x_i-x_j) - \sum_i \frac{-\sum_{k\neq i}(x_i-x_k)^T(x_i-x_k)\exp(-\|x_i-x_k\|_M^2)}{\sum_{k\neq i}\exp(-\|x_i-x_k\|_M^2)}.$$

4.5 Other Distance Metric Learning Techniques

In this section we will study some different proposals for distance metric learning techniques. The algorithms we will analyze are LSI (Xing et al., 2003), DML-eig (Ying and Li, 2012) and LDML (Guillaumin et al., 2009).

4.5.1 LSI

LSI (*learning with side information*) (Xing et al., 2003), also sometimes referred to as MMC (*Mahalanobis metric for clustering*) is a distance metric learning technique that works with a dataset that is not necessarily labeled, which contains certain pairs of samples that are known to be similar and, optionally, pairs of samples that are known not to be similar. It is possibly one of the first algorithms that has helped make the concept of distance metric learning more well known.

LSI tries to learn a metric M that respects this additional information. This is why it can be used both in supervised learning, where similar pairs will correspond to data with the same label, and in unsupervised learning with similarity constraints, such as, for example, clustering problems where it is known that certain samples must be grouped in the same cluster.

We now formulate the problem to be optimized by LSI. Suppose we have the dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, and we know additionally the set $S = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \colon x_i \textit{ and } x_j \textit{ are similar.}\}$. In addition, we may know the set $D = \{(x_i, x_j) \in \mathcal{X} \times \mathcal{X} \colon x_i \textit{ and } x_j \textit{ are dissimilar.}\}$. If we do not have the latter, we can take D as the complement of S in $\mathcal{X} \times \mathcal{X}$.

The first intuition to address this problem, given the information we have, is to minimize the distances between pairs of similar points, that is, to minimize $\sum_{(x_i,x_j)\in S} \|x_i - x_j\|_M^2$, where $M \in S_d(\mathbb{R})_0^+$. However, this will lead us to the solution $M = 0$, which would not give us any productive information. That is why LSI adds the additional constraint $\sum_{(x_i,x_j)\in D} \|x_i - x_j\|_M \geq 1$, which leads us to the optimization problem

$$\begin{aligned} \min_M \quad & \sum_{(x_i,x_j)\in S} \|x_i - x_j\|_M^2 \\ \text{s.t.:} \quad & \sum_{(x_i,x_j)\in D} \|x_i - x_j\|_M \geq 1 \\ & M \in S_d(\mathbb{R})_0^+. \end{aligned}$$

Note several observations regarding this formula. First, the choice of constant 1 in the constraint is irrelevant; if we choose any constant $c > 0$ we get a metric proportional to M. Secondly, the optimization problem is convex, because the sets determined by the restrictions are convex and the function to optimize is also convex. Finally, we may consider a restriction on the set D of the form $\sum_{(x_i,x_j)\in D} \|x_i - x_j\|_M^2 \geq 1$. However, it is possible to rewrite that problem into a formulation similar to that used on the 2-class LDA, where the metric learned would have a rank of 1, which may not be optimal.

To easily optimize this problem, authors propose the equivalent problem

$$\begin{aligned} \max_M \quad & \sum_{(x_i,x_j)\in D} \|x_i - x_j\|_M \\ \text{s.a.:} \quad & \sum_{(x_i,x_j)\in S} \|x_i - x_j\|_M^2 \leq 1 \\ & M \in S_d(\mathbb{R})_0^+. \end{aligned} \tag{11}$$

This problem with two convex constraints can be solved by a projected gradient ascent method. In this problem, constraints are easy to satisfy separately. The first constraint consists of a projection onto an affine half-space, while the second constraint consists of a projection onto the positive semidefinite cone. The method of iterated projections makes it possible to fulfill both restrictions by repeteadly projecting onto both sets until convergence is obtained.

4.5.2 DML-eig

DML-eig (*distance metric learning with eigenvalue optimization*) (Ying and Li, 2012) is a distance metric learning algorithm inspired by the LSI algorithm of the previous section, proposing a very similar optimization problem but offering a completely different resolution method, based on eigenvalue optimization.

We consider, as in the previous case, a training dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, for which we know two sets of pairs, S and D, of data considered similar and dissimilar, respectively. In the previous section, in order to optimize Eq. 11 an ascending gradient method with iterated projections was proposed, which may take a long time to converge. DML-eig proposal consists of a slight modification of the objective function, keeping the same constraints, which leads us to the problem

$$\begin{aligned} \max_M \quad & \min_{(x_i,x_j)\in D} \|x_i - x_j\|_M^2 \\ \text{s.t.:} \quad & \sum_{(x_i,x_j)\in S} \|x_i - x_j\|_M^2 \leq 1 \\ & M \in S_d(\mathbb{R})_0^+. \end{aligned} \tag{12}$$

To address this problem, it is useful to introduce a notation that simplifies the indexing of the data. First, we will denote $X_{ij} = (x_i - x_j)(x_i - x_j)^T$ to the outer products between the differences of the elements in $\mathcal{X}$. To access pairs of elements (i, j) we will use a single index $\tau \equiv (i, j)$. This index can be assumed ordered when necessary, to access the components of a vector of appropiate size. The previous outer product X_{ij} can also be written as X_τ. Finally, for sets S and D, we also assume that they are made by indexes τ associated with a pair (i, j) such that x_i and x_j are similar or dissimilar, respectively. Thus, if we denote $X_S = \sum_{(i,j)\in S} X_{ij}$, Eq. 12 can be rewritten in terms of Frobenius dot product as

$$\begin{aligned} \max_M \quad & \min_{\tau\in D} \langle X_\tau, M\rangle \\ \text{s.t.:} \quad & \langle X_S, M\rangle \leq 1 \\ & M \in S_d(\mathbb{R})_0^+. \end{aligned} \tag{13}$$

Let us see how the formulation of the problem we are looking for is established in terms of eigenvalue optimization. For each symmetric matrix $X \in S_d(\mathbb{R})$ we denote its highest eigenvalue as $\lambda_{\max}(X)$. Associated with the set D of dissimilar pairs we will define the simplex

$$\Delta = \left\{ u \in \mathbb{R}^{|D|} \colon u_\tau \geq 0 \ \forall \tau \in D, \sum_{\tau\in D} u_\tau = 1 \right\}.$$

We also consider the set

$$\mathcal{P} = \{ M \in \mathcal{M}_d(\mathbb{R})_0^+ \colon \operatorname{tr}(M) = 1 \}.$$

$\mathcal{P}$ is the intersection of the positive semidefinite cone with an affine subspace of $\mathcal{M}_d(\mathbb{R})$. Sets with this structure are known as *spectrahedra.*

So, if X_S is positive semidefinite, and we define, for each $\tau \in D$, $\widetilde{X}_\tau = X_S^{-1/2} X_\tau X_S^{-1/2}$, we can prove (see Ying and Li, 2012) that the problem given by Eq. 13 is equivalent to the following problem:

$$\max_{S\in\mathcal{P}} \min_{u\in\Delta} \sum_{\tau\in D} u_\tau \langle \widetilde{X}_\tau, S\rangle,$$

which in turn can be rewritten as an eigenvalue optimization problem:

$$\min_{u\in\Delta}\max_{S\in\mathcal{P}}\left\langle \sum_{\tau\in D} u_\tau \widetilde{X}_\tau, S\right\rangle = \min_{u\in\Delta}\lambda_{\max}\left(\sum_{\tau\in D} u_\tau \widetilde{X}_\tau\right). \tag{14}$$

The problem of minimizing the largest eigenvalue of a symmetric matrix is well-known and there are some iterative methods that allow this minimum to be reached (see Overton, 1988). Furthermore, Ying and Li (2012) also propose an algorithm to solve the problem $\max_{S\in\mathcal{P}}\min_{u\in\Delta}\sum_{\tau\in D}u_\tau\langle\widetilde{X}_\tau, S\rangle + \mu\sum_{\tau\in D}u_\tau\log u_\tau$, where $\mu > 0$ is a smoothing parameter, by means of which the problem in Eq. 14 can be approximated.

4.5.3 LDML

LDML (*logistic discriminant metric learning*) (Guillaumin et al., 2009) is a distance metric learning algorithm in which the optimization model makes use of the logistic function. Authors affirm that this technique is quite useful to learn distances on sets of labeled images, being able to be used therefore in problems like face identification.

Recall that the *logistic* or *sigmoid* function is the map $\sigma\colon \mathbb{R}\to\mathbb{R}$ given by

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

This function presents a graph with a sigmoidal shape, is differentiable, strictly increasing and takes values between 0 and 1, reaching these values in their limits at infinity. These properties allow the logistic function to be the cumulative distribution function of a random variable, which gives it an important probabilistic utility. Its graph presents an asymptotic behaviour from small values (in absolute value), with an exponential growth in zones close to zero. This makes logistic function very useful for modeling binary signals. It also presents a derivative that is easy to calculate, and can be expressed in terms of the logistic function itself, $\sigma'(x) = \sigma(x)(1-\sigma(x))$.

Suppose we have the dataset $\mathcal{X} = \{x_1,\dots,x_N\}\subset\mathbb{R}^d$, with corresponding labels $y_1,\dots,y_N$. In LDML, logistic function is used to define a probability, which will assign the greater probability the smaller the distance between points. To measure the distance, LDML will use a positive semidefinite matrix, resulting in the expression of the probability as

$$p_{ij,M} = \sigma(b - \|x_i - x_j\|_M^2),$$

where b is a positive threshold value that will determine the maximum value achievable by the logistic function, and that can be estimated by cross validation. Associated with this probability, we can define a random variable that follows a Bernouilli distribution, and that takes the values 0 and 1, according to whether the pair (x_i, x_j) belongs to the same class. This distribution is determined by the probability mass function

$$f_{ij,M}(x) = (p_{ij,M})^x(1-p_{ij,M})^{1-x}, \quad x\in\{0,1\}.$$

The function that LDML tries to maximize is the log-likelihood of the previous distribution for the given dataset, that is,

$$\mathcal{L}(M) = \sum_{i,j=1}^{N} y_{ij}\log p_{ij,M} + (1-y_{ij})\log(1-p_{ij,M}),$$

where y_{ij} is a binary variable that takes the value 1 if $y_i = y_j$ and 0 otherwise. This function is differentiable and concave (it is a positive combination of functions that can be expressed as a minus log-sum-exp function, which is concave), so we have a convex maximization problem. Keeping in mind the properties of the logistic function, if $x_{ij} \equiv (x_i - x_j)(x_i - x_j)^T$ and $p_{ij} \equiv p_{ij,M}$, the gradient has the expression

$$\begin{aligned}\nabla \mathcal{L}(M) &= \sum_{i,j=1}^{N} y_{ij} \frac{-x_{ij} p_{ij}(1-p_{ij})}{p_{ij}} + (1-y_{ij}) \frac{x_{ij} p_{ij}(1-p_{ij})}{1-p_{ij}} \\ &= \sum_{i,j=1}^{N} -y_{ij} x_{ij}(1-p_{ij}) + (1-y_{ij}) x_{ij} p_{ij} \\ &= \sum_{i,j=1}^{N} x_{ij}((1-y_{ij})p_{ij} - (1-p_{ij})y_{ij}) \\ &= \sum_{i,j=1}^{N} x_{ij}(p_{ij} - y_{ij}),\end{aligned}$$

The projected gradient method with projections onto the positive semidefinite cone is the semidefinite programming algorithm that is used in LDML to obtain the metric that optimizes its objective function.

4.6 Kernel Distance Metric Learning

In this part we will analyze some of the kernelized versions of the algorithms presented throughout this section. First, we will see how the kernel trick is applied in distance metric learning, and then we will study the kernel algorithms for LMNN, ANMM, DMLMJ and LDA.

4.6.1 The Kernel Trick for Distance Metric Learning

Kernel methods constitute a paradigm within machine learning that is very useful in many of the problems addressed in this discipline. They usually arise in problems where the learning algorithm capability is reduced, typically due to the shape of the dataset. A classic learning algorithm where the kernel trick is very useful is the *support vector machines* classifier (Burges, 1998). An example for this case is given in Figure 12.

In distance metric learning, the usefulness of kernel learning is due to the limitations given by the Mahalanobis distances. Although learned metrics can later be used with non-linear classifiers, such as the nearest neighbors classifier, the metrics themselves are determined by linear transformations, which, in turn, are determined by the image of a basis in the departure space, which results in the fact that we only have the freedom to choose the image of as many data as the dimension has the space, mapping the rest of the vectors by linearity. When the amount of data is much larger than the space dimension this can become a limitation.

The kernel approach for distance metric learning follows a similar scheme to that of support vector machines. If we work with a dataset $\mathcal{X} = \{x_1, \dots, x_N\} \subset \mathbb{R}^d$, the idea is

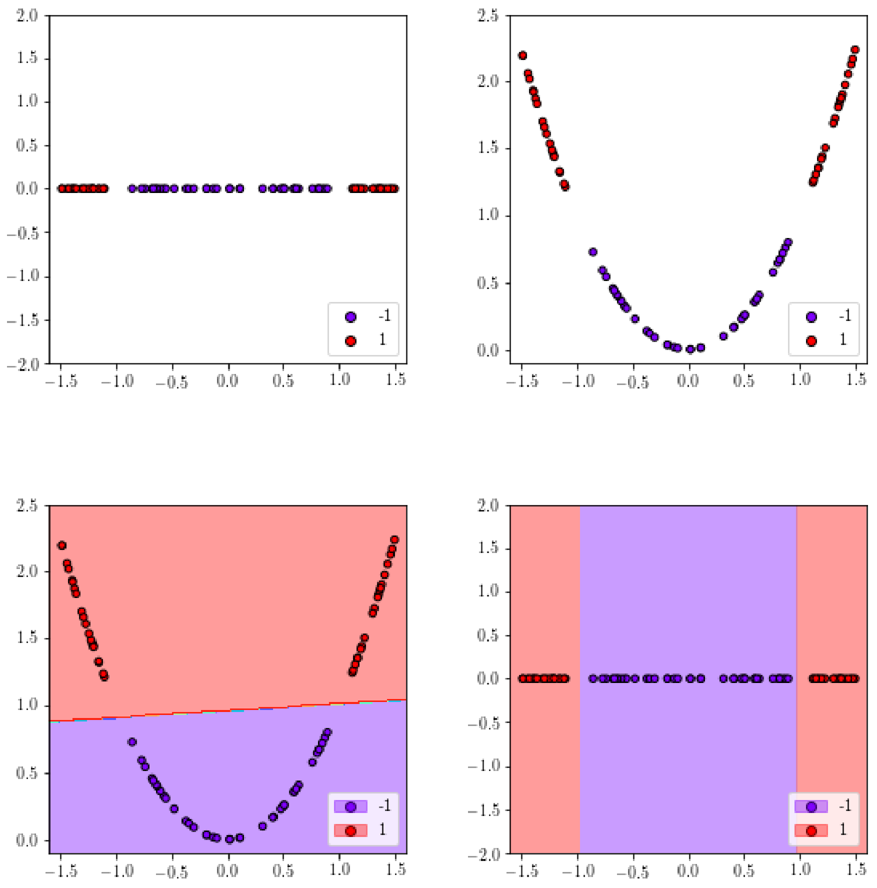

Figure 12: Support vector machines and kernel trick. This binary classifier looks for the hyperplane that best separates both classes. Therefore, it is highly limited when the dataset is not separable by hyperplanes, as in the dataset of the upper-left image. A solution consists of sending the data into a higher dimensional space, where data can be separated by hyperplanes, and apply there the algorithm, as it is shown in the remaining images. The kernel trick allows us to execute the algorithm only in terms of the dot products of the samples in the new space, which makes it possible to work on very high dimensional spaces, or even infinite dimensional spaces. The existence of a *representer theorem* for support vector machines also allows the solution to be rewritten in terms of a vector with the size of the number of samples.

to send the data to a higher dimensional space, through a mapping $\phi\colon \mathbb{R}^d \to \mathcal{F}$, where $\mathcal{F}$ is a Hilbert space called the *feature space*, and then to learn in the feature space using a distance metric learning algorithm. The way we will learn a distance in the feature space will be via a continuous linear transformation $L\colon \mathcal{F} \to \mathbb{R}^{d'}$, where $d' \leq d$ (observe that L is not necessarily a matrix, since $\mathcal{F}$ is not necessarily finite dimensional), which we will also denote $L \in \mathcal{L}(\mathcal{F}, \mathbb{R}^{d'})$.

As occurs with support vector machines, a great inconvenience arises when sending the data to the feature space, and that is that the problem dimension can highly increase, and therefore the application of the algorithms can be very expensive computationally. In addition, if we want to work in infinite dimensional feature spaces, it is impossible to deal with the data in this case, unless we turn to the kernel trick.

We define the *kernel function* as the mapping $K\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ given by $K(x, x') = \langle \phi(x), \phi(x') \rangle$. The success of kernel functions is due to the fact that many learning algorithms only need to know the dot products between the elements in the training set to be able to work. This will happen in the distance metric learning algorithms we will study later. We can observe, as an example, that the calculation of euclidean distances, which is essential in many distance metric learning algorithms, can be made using only the kernel function. Indeed, for $x, x' \in \mathbb{R}^d$, we have

$$\begin{aligned}
\|\phi(x) - \phi(x')\|^2 &= \langle \phi(x) - \phi(x'), \phi(x) - \phi(x') \rangle \\
&= \langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \phi(x') \rangle + \langle \phi(x'), \phi(x') \rangle \\
&= K(x, x) + K(x', x') - 2K(x, x').
\end{aligned} \tag{15}$$

The next common problem for all the kernel-based distance metric learning algorithms is how to deal with the learned transformation. Since we are trying to learn a map $L \in \mathcal{L}(\mathcal{F}, \mathbb{R}^{d'})$, we may not be able to write it as a matrix, and when we can, this matrix may have dimensions that are too large. However, as L is continuous and linear, using the Riesz representation theorem, we can rewrite L as a vector of dot products by fixed vectors, that is, $L = (\langle \cdot, w_1 \rangle, \dots, \langle \cdot, w_{d'} \rangle)$, where $w_1, \dots, w_{d'} \in \mathcal{F}$. Furthermore, for the algorithms we will study, several *representer theorems* are known (see Chatpatanasiri et al., 2010; Hofmann et al., 2008; Mika et al., 1999; Nguyen et al., 2017; Schölkopf et al., 1998). These theorems allow the vectors w_i to be expressed as a linear combination of the samples in the feature space, that is, for each $i \in \{1, \dots, d'\}$, there is a vector $\alpha^i = (\alpha^i_1, \dots, \alpha^i_N) \in \mathbb{R}^N$ so that $w_i = \sum_{j=1}^N \alpha^i_j \phi(x_j)$. Consequently, we can see that

$$L\phi(x) = A \begin{pmatrix} K(x_1, x) \\ \vdots \\ K(x_N, x) \end{pmatrix}, \tag{16}$$

where $A \in \mathcal{M}_{d' \times N}(\mathbb{R})$ is given by $A_{ij} = \alpha^i_j$.

Thanks to these theorems, we can address the problem computationally as long as we are able to calculate the coefficients of matrix A. When transforming a new sample it will be enough to construct the previous column matrix evaluating the kernel function between the sample and each element in the training set, and then multiplying A by this matrix. On a final note, when training it is useful to view the kernel map as a matrix $K \in S_N(\mathbb{R})$, where

$K_{ij} = K(x_i, x_j)$. A similar (in this case not necessarily square) matrix can be constructed when testing, with all the dot products between the train and test samples. Choosing the appropiate column of this matrix, we will be able to transform the corresponding test sample using Eq. 16.

Each distance metric learning technique that supports the use of kernels will use different tools for its performance, each one based on the original algorithms. The following subsections will describe the kernelizations of some of the algorithms already studied.

4.6.2 KLMNN

KLMNN (Torresani and Lee, 2007; Weinberger and Saul, 2009) is the kernelized version of LMNN. In it, the data in $\mathcal{X}$ is sent to the feature space to learn in that space a distance that minimizes the objective function set in the LMNN problem.

Although the problem formulated in the non-kernelized version was made with respect to a positive semidefinite matrix M, using the error function given in Eq. 7, when working in feature spaces we are more interested in dealing with a linear map, even if the convexity of the problem is lost, in order to be able to use the representer theorem. Therefore, adapting the error function proposed in Eq. 6 to the feature space, the LMNN problem for the kernelized version consists of

$$\min_{L \in \mathcal{L}(\mathcal{F}, \mathbb{R}^d)} \quad \varepsilon(L) = (1-\mu)\sum_{i=1}^{N}\sum_{j \rightsquigarrow i}\|L(\phi(x_i) - \phi(x_j))\|^2 \\ + \mu \sum_{i=1}^{N}\sum_{j \rightsquigarrow i}\sum_{l=1}^{N}(1-y_{il})[1 + \|L(\phi(x_i)-\phi(x_j))\|^2 - \|L(\phi(x_i)-\phi(x_l))\|^2]_+.$$

As a consequence of the representer theorem, it follows that, for each $x_i \in \mathcal{X}$, $L\phi(x) = AK_{.i}$, where $A \in \mathcal{M}_{d' \times N}(\mathbb{R})$ is the matrix given by the representer theorem, and $K_{.i}$ represents the i-th column of the kernel matrix for the training set. Using this in the error expression, we obtain

$$\begin{aligned} &(1-\mu)\sum_{i=1}^{N}\sum_{j \rightsquigarrow i}\|L(\phi(x_i) - \phi(x_j))\|^2 \\ &\quad + \mu \sum_{i=1}^{N}\sum_{j \rightsquigarrow i}\sum_{l=1}^{N}(1-y_{il})[1 + \|L(\phi(x_i)-\phi(x_j))\|^2 - \|L(\phi(x_i)-\phi(x_l))\|^2]_+ \\ &= (1-\mu)\sum_{i=1}^{N}\sum_{j \rightsquigarrow i}\|A(K_{.i} - K_{.j})\|^2 \\ &\quad + \mu \sum_{i=1}^{N}\sum_{j \rightsquigarrow i}\sum_{l=1}^{N}(1-y_{il})[1 + \|A(K_{.i}-K_{.j})\|^2 - \|A(K_{.i}-K_{.l})\|^2]_+. \end{aligned}$$

The above expression depends only on A and kernel functions, and minimizing it as a function of A (we will denote it $\varepsilon(A)$) we get the same value as minimizing $\varepsilon(L)$. Note also that the expression $\varepsilon(A)$ also requires the calculation of target neighbors and impostors, but these depend only on the distances in the feature space, which, as we have already seen, are computable, as shown in Eq. 15. Therefore, all the components of $\varepsilon(A)$ are computationally manipulable, so if we apply a gradient descent method on $\varepsilon(A)$ we can reduce the value of the objective function, always keeping in mind that we can be stuck in a local optimum, because the problem is not convex. Finally, once a matrix A that minimizes $\varepsilon(A)$ is found, we will have determined the corresponding map L thanks to the representer theorem, and we can use A together with the kernel functions to transform new data.

4.6.3 KANMM

KANMM (Wang and Zhang, 2007) is the kernelized version of ANMM. In it, the data in $\mathcal{X}$ is sent to the feature space via the map $\phi\colon \mathbb{R}^d \to \mathcal{F}$, where ANMM is applied to obtain the linear map we are looking for.

Recall that the first step for the application of ANMM was to obtain the homogeneous and heterogeneous neighborhoods for each sample $x_i \in \mathcal{X}$. Note that for this calculation it is only necessary to compare distances in the feature space, which we have seen can be done thanks to the kernel function, through Eq. 15. We will denote the neighborhoods in the feature space as $N^o_{\phi(x_i)}$ y $N^e_{\phi(x_i)}$, respectively, for each x_i.

The scatter and compactness matrices (or endomorphisms, more in general) in the feature space are given by

$$S^\phi = \sum_{i,k\colon \phi(x_k)\in N^e_{\phi(x_i)}} \frac{(\phi(x_i)-\phi(x_k))(\phi(x_i)-\phi(x_k))^T}{|N^e_{\phi(x_i)}|}$$

$$C^\phi = \sum_{i,j\colon \phi(x_j)\in N^o_{\phi(x_i)}} \frac{(\phi(x_i)-\phi(x_j))(\phi(x_i)-\phi(x_j))^T}{|N^o_{\phi(x_i)}|}.$$

The problem to be optimized is therefore expressed as

$$\begin{aligned} \max_{L\in\mathcal{L}(\mathcal{F},\mathbb{R}^{d'})} \quad & \operatorname{tr}\left(L(S^\phi - C^\phi)L^T\right) \\ \text{s.t.:} \quad & LL^T = I. \end{aligned} \tag{17}$$

According to the representer theorem, $L\varphi(x_i) = AK_{.i}$, where A is the matrix of coefficients of the representation theorem and $K_{.i}$ represents the i-th column of the kernel matrix for the training set. Then,

$$L(\phi(x_i)-\phi(x_j))(\phi(x_i)-\phi(x_j))^T L^T = A(K_{.i}-K_{.j})(K_{.i}-K_{.j})^T A^T,$$

and if we consider the matrices

$$\widetilde{S}^\phi = \sum_{i,k\colon \phi(x_k)\in N^e_{\phi(x_i)}} \frac{(K_{.i}-K_{.k})(K_{.i}-K_{.k})^T}{|N^e_{\phi(x_i)}|}$$

$$\widetilde{C}^\phi = \sum_{i,j\colon \phi(x_j)\in N^o_{\phi(x_i)}} \frac{(K_{.i}-K_{.j})(K_{.i}-K_{.j})^T}{|N^o_{\phi(x_i)}|},$$

it follows that the average neighborhood margin is given by

$$\gamma^L = \operatorname{tr}(L(S^\phi - C^\phi)L^T) = \operatorname{tr}(LS^\phi L^T - LC^\phi L^T) = \operatorname{tr}(A\widetilde{S}^\phi A^T - A\widetilde{C}^\phi A^T = \operatorname{tr}(A(\widetilde{S}^\phi - \widetilde{C}^\phi)A^T)).$$

If we impose the restriction $AA^T = I$, Theorem 11 tells us again that we can take matrix A that which contains as rows the eigenvectors of $\widetilde{S}^\phi - \widetilde{C}^\phi$ corresponding to its d' largest eigenvalues. Observe that we can calculate both matrices from the kernel function, and the matrix A we obtain determines the linear map, as a consequence of the representer theorem. Therefore, we have finally obtained a kernel-based method for applying ANMM in feature spaces.

4.6.4 KDMLMJ

KDMLMJ (Nguyen et al., 2017) is the kernelized version of DMLMJ. In it, the data in $\mathcal{X}$ is sent to the feature space, where a distance is learned after applying DMLMJ.

Again, it is possible to calculate the k-positive and k-negative neighborhoods, $V_k^+(\phi(x_i))$ and $V_k^-(\phi(x_i))$, for each $x_i \in \mathcal{X}$, thanks to Eq. 15. It is not the same with the endomorphisms associated with the difference spaces,

$$\Sigma_S^\phi = \frac{1}{|S|}\sum_{i=1}^N \left[\sum_{\phi(x_j)\in V_k^+(\phi(x_i))} (\phi(x_i) - \phi(x_j))(\phi(x_i) - \phi(x_j))^T\right]$$
$$\Sigma_D^\phi = \frac{1}{|D|}\sum_{i=1}^N \left[\sum_{\phi(x_j)\in V_k^-(\phi(x_i))} (\phi(x_i) - \phi(x_j))(\phi(x_i) - \phi(x_j))^T\right].$$

The optimization problem is given by

$$\max_{L\in\mathcal{L}(\mathcal{F},\mathbb{R}^{d'})} \quad J(L) = \operatorname{tr}\left((L\Sigma_S^\phi L^T)^{-1}(L\Sigma_D^\phi L^T) + (L\Sigma_D^\phi L^T)^{-1}(L\Sigma_S^\phi L^T)\right).$$

Again we have, as a consequence of the representer theorem, that $L\phi(x_i) = AK_{.i}$ for each $x_i \in \mathcal{X}$, where A is the matrix provided by the representer theorem, and $K_{.i}$ is the i-th column of the kernel matrix for the training set. If, reasoning as in the previous section, we define the matrices

$$U = \frac{1}{|S|}\sum_{i=1}^N \left[\sum_{\phi(x_j)\in V_k^+(\phi(x_i))} (K_{.i} - K_{.j})(K_{.i} - K_{.j})^T\right]$$
$$V = \frac{1}{|D|}\sum_{i=1}^N \left[\sum_{\phi(x_j)\in V_k^-(\phi(x_i))} (K_{.i} - K_{.j})(K_{.i} - K_{.j})^T\right],$$

we obtain that

$$\operatorname{tr}\left((L\Sigma_S^\phi L^T)^{-1}(L\Sigma_D^\phi L^T) + (L\Sigma_D^\phi L^T)^{-1}(L\Sigma_S^\phi L^T)\right) =$$
$$\operatorname{tr}\left((AUA^T)^{-1}(AVA^T) + (AVA^T)^{-1}(AUA^T)\right).$$

As with DMLMJ, Theorem 13 tells us that we can find a matrix A that maximizes this last equality by taking the eigenvectors of $U^{-1}V$ for which the value $\lambda + 1/\lambda$ is maximized, where λ is the associated eigenvalue. As matrices U and V can be obtained from the kernel function, and A determines L by the representer theorem, we have obtained an algorithm for the application of DMLMJ in the feature space.

4.6.5 KDA

KDA (*kernel discriminant analysis*) (Mika et al., 1999) is the kernelized version of linear discriminant analysis. The kernelization of this algorithm will make it possible to find non-linear directions that nicely separate the data according to the criteria established in the discriminant analysis. Once again, we send the data in $\mathcal{X}$ to the feature space using the mapping $\phi\colon \mathbb{R}^d \to \mathcal{F}$. On that space we will apply linear discriminant analysis.

Suppose, as in LDA, that the set of possible classes is $\mathcal{C}$, of cardinal r, and for each $c \in \mathcal{C}$ we define $\mathcal{C}_c = \{i \in \{1, \dots, N\}\colon y_i = c\}$ and $N_c = |\mathcal{C}_c|$, with μ_c^ϕ the mean vector of the class c, and μ^ϕ the mean vector of the whole dataset, considering it within the feature space. The problem we want to solve in this case is

$$\max_{L \in \mathcal{L}(\mathcal{F}, \mathbb{R}^{d'})} \quad \operatorname{tr}\left((LS_w^\phi L^T)^{-1}(LS_b^\phi L^T)\right), \tag{18}$$

where S_b^ϕ and S_w^ϕ are the operators that measure the between-class and within-class scatter, respectively, and are given by

$$\begin{aligned} S_b^\phi &= \sum_{c \in \mathcal{C}} (\mu_c^\phi - \mu^\phi)(\mu_c^\phi - \mu^\phi)^T \\ S_w^\phi &= \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{C}_c} (\phi(x_i) - \mu_c^\phi)(\phi(x_i) - \mu_c^\phi)^T. \end{aligned}$$

Again, we use the representer theorem, so that if $L \in \mathcal{L}(\mathcal{F}, \mathbb{R}^{d'})$, then, for each $x \in \mathbb{R}^d$,

$$L\phi(x) = A \begin{pmatrix} K(x_1, x) \\ \vdots \\ K(x_N, x) \end{pmatrix},$$

where A is in the conditions of the representer theorem. Let us look again for an expression of the problem given in Eq. 18 that depends only on the kernel function and the matrix A. To do this, we have to observe that for the mean vectors of each class we have

$$L\mu_c^\phi = L\left(\frac{1}{N_c}\sum_{i \in \mathcal{C}_c} \phi(x_i)\right) = \frac{1}{N_c}\sum_{i \in \mathcal{C}_c} L\phi(x_i) = \frac{1}{N_c}\sum_{i \in \mathcal{C}_c} AK_{.i},$$

where $K_{.i}$ is the i-th column of the kernel matrix for the training set. Similarly, for the global mean vector, we have

$$L\mu^\phi = \frac{1}{N}\sum_{i=1}^{N} AK_{.i}.$$

Consequently,

$$\begin{aligned}L(\mu_c^\phi - \mu^\phi)(\mu_c^\phi - \mu^\phi)^T L^T &= (L\mu_c^\phi - L\mu^\phi)(L\mu_c^\phi - L\mu^\phi)^T \\ &= \left(\frac{1}{N_c}\sum_{i\in\mathcal{C}_c} AK_{.i} - \frac{1}{N}\sum_{i=1}^N AK_{.i}\right)\left(\frac{1}{N_c}\sum_{i\in\mathcal{C}_c} AK_{.i} - \frac{1}{N}\sum_{i=1}^N AK_{.i}\right)^T.\end{aligned}$$

Note that the last expression depends only on A and the kernel function. Moreover, for $x_i \in \mathcal{X}$ with $y_i = c$, we have

$$\begin{aligned}L(\phi(x_i) - \mu_c^\phi)(\phi(x_i) - \mu_c^\phi)^T L^T &= (L\phi(x_i) - L\mu_c^\phi)(L\phi(x_i) - L\mu_c^\phi)^T \\ &= \left(AK_{.i} - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.j}\right)\left(AK_{.i} - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.j}\right)^T \\ &= \left(AK_{.i} - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.j}\right)\left(K_{.i}^T A^T - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} K_{.j}^T A^T\right) \\ &= AK_{.i}K_{.i}^T A^T - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.i}K_{.j}^T A^T - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.j}K_{.i}^T A^T + \frac{1}{N_c^2}\sum_{j\in\mathcal{C}_c}\sum_{l\in\mathcal{C}_c} AK_{.j}K_{.l}^T A^T.\end{aligned}$$

By summing in $i \in \mathcal{C}_c$, we obtain

$$\begin{aligned}&\sum_{i\in\mathcal{C}_c} L(\phi(x_i) - \mu_c^\phi)(\phi(x_i) - \mu_c^\phi)^T L^T \\ &= \sum_{i\in\mathcal{C}_c}\left[AK_{.i}K_{.i}^T A^T - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.i}K_{.j}^T A^T - \frac{1}{N_c}\sum_{j\in\mathcal{C}_c} AK_{.j}K_{.i}^T A^T + \frac{1}{N_c^2}\sum_{j\in\mathcal{C}_c}\sum_{l\in\mathcal{C}_c} AK_{.j}K_{.l}^T A^T\right] \\ &= \sum_{i\in\mathcal{C}_c} AK_{.i}K_{.i}^T A^T - \frac{2}{N_c}\sum_{i\in\mathcal{C}_c}\sum_{j\in\mathcal{C}_c} AK_{.i}K_{.j}^T A^T + \frac{1}{N_c^2}\sum_{i\in\mathcal{C}_c}\sum_{j\in\mathcal{C}_c}\sum_{l\in\mathcal{C}_c} AK_{.j}K_{.l}^T A^T \\ &= \sum_{i\in\mathcal{C}_c} AK_{.i}K_{.i}^T A^T - \frac{2}{N_c}\sum_{i\in\mathcal{C}_c}\sum_{j\in\mathcal{C}_c} AK_{.i}K_{.j}^T A^T + \frac{N_c}{N_c^2}\sum_{j\in\mathcal{C}_c}\sum_{l\in\mathcal{C}_c} AK_{.j}K_{.l}^T A^T \\ &= \sum_{i\in\mathcal{C}_c} AK_{.i}K_{.i}^T A^T - \frac{1}{N_c}\sum_{i\in\mathcal{C}_c}\sum_{j\in\mathcal{C}_c} AK_{.i}K_{.j}^T A^T \\ &= AK_c K_c^T A^T - AK_c\left(\frac{1}{N_c}\mathbb{1}\right)K_c^T A^T \\ &= AK_c\left(I - \frac{1}{N_c}\mathbb{1}\right)K_c^T A^T,\end{aligned}$$

where $\mathbb{1} \in \mathcal{M}_{N_c}(\mathbb{R})$ is a square matrix with the value 1 in all its entries, and $K_c \in \mathcal{M}_{N\times N_c}$ is a kernel matrix whose entries are the values of the kernel function between all the samples in $\mathcal{X}$ and the samples with class c. Again, this last expression depends only on A and the kernel function.

If we finally define

$$\begin{aligned}
U_c &= \frac{1}{N_c}\sum_{i\in\mathcal{C}_c} K_{.i} \in \mathbb{R}^N, c \in \mathcal{C} \\
U_\mu &= \frac{1}{N}\sum_{j=1}^{N} K_{.i} \in \mathbb{R}^N \\
U &= \sum_{c\in\mathcal{C}} N_c(U_c - U_\mu)(U_c - U_\mu)^T \in S_N(\mathbb{R}) \\
V &= \sum_{c\in\mathcal{C}} K_c\left(I - \frac{1}{N_c}\mathbb{1}\right)K_c^T \in S_N(\mathbb{R}),
\end{aligned}$$

we can conclude that

$$\operatorname{tr}\left((LS_w^\phi L^T)^{-1}(LS_b^\phi L^T)\right) = \operatorname{tr}\left((AVA^T)^{-1}(AUA^T)\right),$$

where U and V are computable using the kernel function. Therefore, we obtain a problem equivalent to the original given in Eq. 18, but in terms of A, for which Theorem 12 states that, if U is positive definite, we can maximize the value of the trace by taking as rows of A the eigenvectors of $V^{-1}U$ corresponding to its d' largest eigenvalues. In this way, since A determines L thanks to the representer theorem, we obtain a kernel-based method for the application of discriminant analysis in feature spaces.

5. Software Libraries

In this section, we will introduce the distance metric learning software we have developed. Our software package has been developed for the Python language and aims to make a wide variety of distance metric learning algorithms easily available to users. First, we will review the existing software on this topic and then we will describe our software, along with its main features.

5.1 Existing Software

Several distance metric learning software proposals have been developed in different programming languages. The most outstanding proposals have been developed for three of the most popular languages in the field of machine learning: R, MATLAB and Python.

In R, we can find the package `dml`[2]. This package consists of a group of 11 supervised distance metric learning algorithms. However, many of them are not implemented yet; in fact, there are currently only 5 algorithms available. In addition, this package has not exhibited activity for more than a year.

In MATLAB, it is possible to find the `DistLearn`[3] toolkit. This package is a collection of the implementations of several distance metric learning algorithms. Many of these implementations are, in fact, those performed by the authors of the algorithm. We can

2. CRAN page: `https://CRAN.R-project.org/package=dml`; source code: `https://github.com/terrytangyuan/dml`; docs: `https://cran.r-project.org/web/packages/dml/dml.pdf`.
3. `DistLearn` webpage: `https://www.cs.cmu.edu/~liuy/distlearn.htm`.

find in this toolkit up to 9 supervised learning algorithms. In addition, it also includes several unsupervised algorithms. However, this package seems to be out of date since 2007, and many of the links are currently broken. It is also important to note that some of the algorithms presented in the toolkit follow a more oriented approach to manifold learning (Lee and Verleysen, 2007) than to the Mahalanobis distance learning approach studied in this paper.

Finally, in Python, we can find the `metric-learn`[4] library. This library consists of 9 supervised learning algorithms. It includes some of the classic algorithms studied in this tutorial, such as LMNN, NCA, ITML or LSI, and the remaining algorithms are mainly oriented to weak supervised learning, and are different from those studied in this paper.

5.2 pyDML: A Python Library for Distance Metric Learning

We have developed a Python library, called `pyDML`[5], which contains all the 17 algorithms studied in Section 4. Python is a programming language widely used in machine learning, and has several libraries specialized in this field. The main one is `Scikit-Learn`[6] (Pedregosa et al., 2011), an efficient open-source library for machine learning, which relies on the Scipy[7] ecosystem, which contains numerical calculus libraries, such as `NumPy`[8], data processing libraries, such as `Pandas`[9], or data visualization libraries, such as `Matplotlib`[10].

The choice of Python is due to the fact that until now, Python does not have an extensive library with supervised distance learning algorithms. In `Scikit-Learn` it is possible to find algorithms such as PCA or LDA, included in the group of dimensionality reduction techniques, but we cannot find any other distance metric learning algorithm beyond this. We have also seen in the previous section that the existing supervised distance metric learning software is not very elaborate. The `pyDML` library tries to fill these gaps, providing numerous supervised distance learning algorithms, both classic algorithms and new proposals.

The design followed for the development of the algorithms has preserved the structure of the algorithms of the `Scikit-Learn` library. In particular, the distance metric learning algorithms are included in the group of transformation algorithms, where the transformation consists precisely in applying the learned linear map to the samples. Therefore, the algorithms have been implemented as subclasses of the `sklearn.base.TransformerMixin`[11] class of the Scikit-Learn toolkit.

Additionally, distance metric learning algorithms provide a positive semidefinite metric matrix (except kernel-based algorithms) and a linear transformation (both recoverable from each other, although each algorithm learns only one of these options). It is important to be able to access them for distance calculation, so all distance metric learning algorithms will implement, in addition to the methods `fit` and `transform` defined

4. PyPI page: `https://pypi.org/project/metric-learn/`; source code: `https://github.com/metric-learn/metric-learn`; docs: `http://metric-learn.github.io/metric-learn/`.
5. The source code of `pyDML` is available in GitHub: `https://github.com/jlsuarezdiaz/pyDML`.
6. `http://scikit-learn.org/stable/`.
7. `https://www.scipy.org/`.
8. `http://www.numpy.org/`.
9. `http://pandas.pydata.org/`.
10. `https://matplotlib.org/`.
11. `http://scikit-learn.org/stable/modules/generated/sklearn.base.TransformerMixin.html#sklearn.base.TransformerMixin`

in `sklearn.base.TransformerMixin` to learn and transform, respectively, the methods `metric` and `transformer`, which will give access to these elements.

For this reason, an abstract class `DML_Algorithm` has been implemented, which provides the functionalities common to all the distance metric learning algorithms, that is, it allows the learned matrices to be accessed through the methods `metric()` and `transformer()`, and defines the function `transform(X)`, which returns the dataset resulting from applying the learned transformation to each sample in `X`. It is also included a `metadata()` method, which provides additional information about what each algorithm has learned, useful when estimating parameters.

Each of the distance metric learning algorithms implements a subclass of `DML_Algorithm` implementing a method `fit(X,y)`, responsible for learning the distance from the labeled data given by `X` and `y`. For kernel-based algorithms, a subclass of `DML_Algorithm` has been created, which provides the kernel-specific functionalities. Figure 13 describes the classes structure of our software.

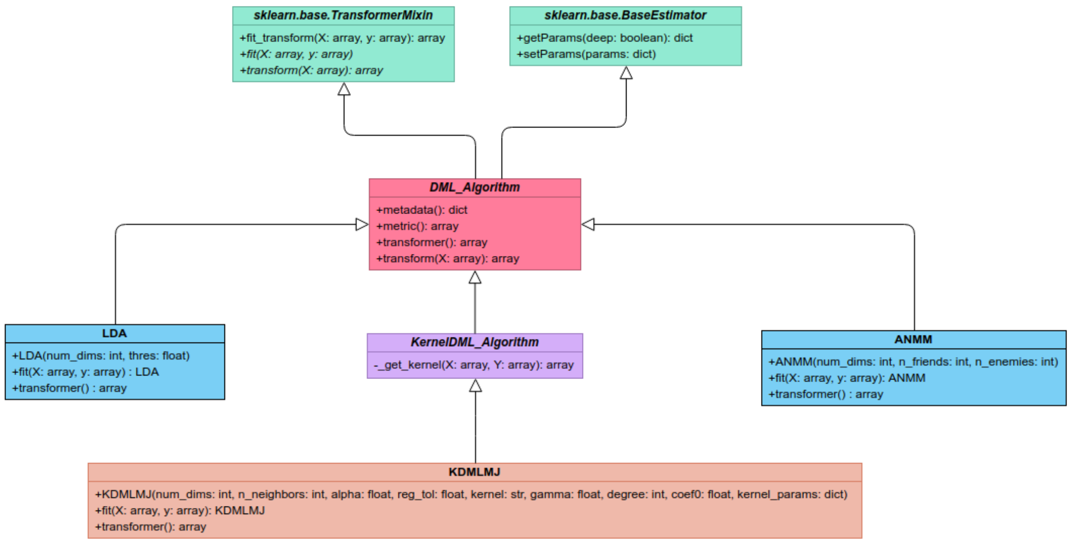
sklearn.base.TransformerMixin
+fit_transform(X: array, y: array): array
+fit(X: array, y: array)
+transform(X: array): array
sklearn.base.BaseEstimator
+getParams(deep: boolean): dict
+setParams(params: dict)
DML_Algorithm
+metadata(): dict
+metric(): array
+transformer(): array
+transform(X: array): array
LDA
+LDA(num_dims: int, thres: float)
+fit(X: array, y: array) : LDA
+transformer() : array
KernelDML_Algorithm
-_get_kernel(X: array, Y: array): array
ANMM
+ANMM(num_dims: int, n_friends: int, n_enemies: int)
+fit(X: array, y: array): ANMM
+transformer() : array
KDMLMJ
+KDMLMJ(num_dims: int, n_neighbors: int, alpha: float, reg_tol: float, kernel: str, gamma: float, degree: int, coef0: float, kernel_params: dict)
+fit(X: array, y: array): KDMLMJ
+transformer(): array

Figure 13: Class diagram of the algorithms developed in `pyDML` (only LDA, ANMM and KANMM are shown, but the rest of the algorithms follow a similar structure, overriding the `metric()` or `transformer()` method, depending on how the algorithm has learned).

It is important to emphasize that these algorithms include different hyperparameters that can be modified to improve the performance ot to change the conditions of the learned distances. These parameters vary with each algorithm, and the most common and of greatest relevance in the execution of the algorithms are:

- **Number of dimensions.** The desired dimension for the transformation to learn. It is present in all dimensionality reduction algorithms, although most of the algorithms that learn linear maps admit this parameter.

- **Number of neighbors.** The parameters of this type determine how many samples are chosen for the construction of neighborhoods in the algorithms that perform these constructions, as is the case for ANMM, LMNN or DMLMJ.
- **Descent method or solver.** In algorithms that minimize a differentiable function, the desired descent method can be established. For unconstrained problems, valid options are currently *batch gradient descent* (BGD) or *stochastic gradient descent* (SGD). For constrained problems over the positive semidefinite cone, the semidefinite programming method (SDP) consisting of projected gradient descent is allowed.
- **Learning rate.** In algorithms based on gradient descent, an initial value for the learning rate can be set, as well as how it varies: constant or adaptive. In the latter case, if the gradient iteration has improved the value of the objective function, the rate is increased. Otherwise, it is reduced. The increase and decrease factors are also parameters that can be set.
- **Stop criteria.** In algorithms based on gradient descent, it is necessary to establish stop criteria. The criteria available in most algorithms consist of maximum number of iterations, precision (closeness of the gradient norm to zero) or tolerance (difference between two iterations in the descent algorithm).
- **Regularization parameters.** Some algorithms require regularization parameters, of different natures depending on the algorithm. Both the regularization parameters and the criteria that determine their application can be set in the algorithms that require it.

A detailed description of all hyperparameters for each algorithm can be found in the `pyDML`'s full documentation[12].

In addition to the learning algorithms, two distance-based classifiers have been developed, also following the `Scikit-Learn` structure. The first one is a wrapper for the nearest neighbors classifier provided in `Scikit-Learn`, to make it easier to act in conjunction with a distance metric learning algorithm, which until now was not possible to do directly with the `Scikit-Learn` nearest neighbors classifier. The second one is the multiple centroid classifier NCMC. `Scikit-Learn` currently has only the nearest class mean classifier[13]. NCMC classifier adds the possibility of using more than one centroid per class, and thus can be optimized by its corresponding distance metric learning algorithm NCMC.

The `pyDML` library also incorporates graphical tools for the representation and evaluation of the learned distances, which use the `Matplotlib` library internally. These tools allow labeled data to be represented, along with the regions determined by any `Scikit-Learn` classifier (`classifier_plot`). They also show how a region determined by a distance-based classifier changes when the distance is changed (`dml_plot`). A specialized function makes it easier to plot regions for the nearest neighbors classifiers (`knn_plot`). Similarly, functions are added to represent different attribute pairs, along with a section of the classifier region, for higher dimensional datasets (`classifier_pairplots`, `dml_pairplots`, `knn_pairplots`).

12. `https://pydml.readthedocs.io/`
13. See `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestCentroid.html`

Finally, multiple plots comparing different classifiers or distances can also be elaborated, thanks to the function `dml_multiplot`. A detailed description of these functions and their arguments can be found again in the documentation[12]. The parameters of the `tune` functions can also be found here. This is the last additional functionality included, which allows the parameters of the distance metric learning algorithms to be easily estimated with cross validation, using the success rate of a k-neighbors classifier or some of the metadata of the algorithms as validation metrics.

The `pyDML` library can be installed through `PyPI` (*Python package index*), using the command `pip install pyDML` in a shell. It is also possible to download or clone the repository directly from GitHub. In such a case, the installation of the software package can be done by running the setup script available in the root directory, using the command `python setup.py install`. Once installed, we can access all distance metric learning algorithms, and the additional functionalities, by importing the desired class within the `dml` module.

As already mentioned, the way distance metric learning algorithms are used is similar to the `Scikit-Learn` transformers. Figure 14 shows an example for the case of the NCA algorithm. With the rest of the algorithms the operation is the same, possibly varying the different hyperparameters.

The use of the classifiers in the library is analogous to the use of `Scikit-Learn` classifiers. For the plotting functions, an example is shown in Figure 15. More detailed examples of all the possibilities offered by `pyDML` can be found in the documentation[14].

6. Experimental Framework and Results

With the `pyDML` library developed and explained in the previous section, several experiments have been carried out with the different algorithms implemented. This section describes these experiments and shows the results.

6.1 Description of the Experiments

For the distance metric learning algorithms studied, a collection of experiments has been developed, consisting of the following procedures.

1. Evaluation of all the algorithms capable of learning at maximum dimension, applied to the k-nearest neighbors classification, for different values of k.

2. Evaluation of the algorithms aimed at improving nearest centroid classifiers, applied to the corresponding centroid-based classifiers.

3. Evaluation of kernel-based algorithms, experimenting with different kernels, applied to the nearest neighbors classification.

4. Evaluation of algorithms capable of reducing dimensionality, for different dimensions, applied to the nearest neighbors classification.

14. `https://pydml.readthedocs.io/en/latest/examples.html`.

```
>>> import numpy as np                      # NumPy library
>>> from sklearn.datasets import load_iris # Iris dataset
>>> from dml import NCA                     # Loading DML algorithm

>>> # Loading dataset
>>> iris = load_iris()
>>> X = iris['data']
>>> y = iris['target']

>>> nca = NCA()  # DML construction
>>> nca.fit(X,y) # Fitting algorithm

>>> # We can look at the algorithm metadata after fitting it
>>> meta = nca.metadata()
>>> meta
{'final_expectance': 0.95771240234375,
 'initial_expectance': 0.8380491129557291,
 'num_iters': 3}

>>> M = nca.metric() # We can see the metric the algorithm has learned.
>>> M
array([[ 1.19098678,  0.51293714, -2.15818151, -2.01464351],
       [ 0.51293714,  1.58128238, -2.14573777, -2.10714773],
       [-2.15818151, -2.14573777,  6.46881853,  5.86280474],
       [-2.01464351, -2.10714773,  5.86280474,  6.83271473]])

>>> L = nca.transformer() # Equivalently, we can see the learned linear map.
>>> L
array([[ 0.77961001, -0.01911998, -0.35862791, -0.23992861],
       [-0.04442949,  1.00747788, -0.29936559, -0.25812144],
       [-0.60744415, -0.57288453,  2.16095076,  1.35212555],
       [-0.46068713, -0.48755353,  1.25732916,  2.20913531]])

# Finally, we can obtain the transformed data, or transform new data.
>>> Lx = nca.transform() # Transforming training set.
>>> Lx[:5,:]
array([[ 3.35902632,  2.8288461 , -1.80730485, -1.85385382],
       [ 3.21266431,  2.33399305, -1.39937375, -1.51793964],
       [ 3.0887811 ,  2.57431109, -1.60855691, -1.64904583],
       [ 2.94100652,  2.41813313, -1.05833389, -1.30275593],
       [ 3.27915332,  2.93403684, -1.80384889, -1.85654046]])
```

Figure 14: Use of distance metric learning algorithms in `pyDML`.

```
>>> import numpy as np                                          # NumPy library
>>> from sklearn.datasets import load_iris                      # Iris dataset
>>> from dml import NCA, LDA, NCMC_Classifier, dml_multiplot # Learning functions
>>> iris = load_iris()
>>> X = iris['data']
>>> y = iris['target']
>>> # Initializing learning algorithms.
>>> nca = NCA()
>>> lda = LDA()
>>> ncmc = NCMC()
>>> # Let's see how the 3NN and NCMC classifiers change after learning a distance.
>>> # We will plot 4 figures: {NCMC, NCMC+NCA, 3-NN, 3-NN+LDA}
>>> dml_multiplot(X[:,[0,1]],y,nrow=2,ncol=2,ks=[None,None,3,3],
>>>               clfs=[ncmc,ncmc,None,None],dmls=[None,nca,None,lda],
>>>               transforms=[False,False,False,False],title="Comparing",
>>>               subtitles=["NCMC","NCMC + NCA","3-NN","3-NN + LDA"],
>>>               cmap="rainbow",figsize=(12,12))
```

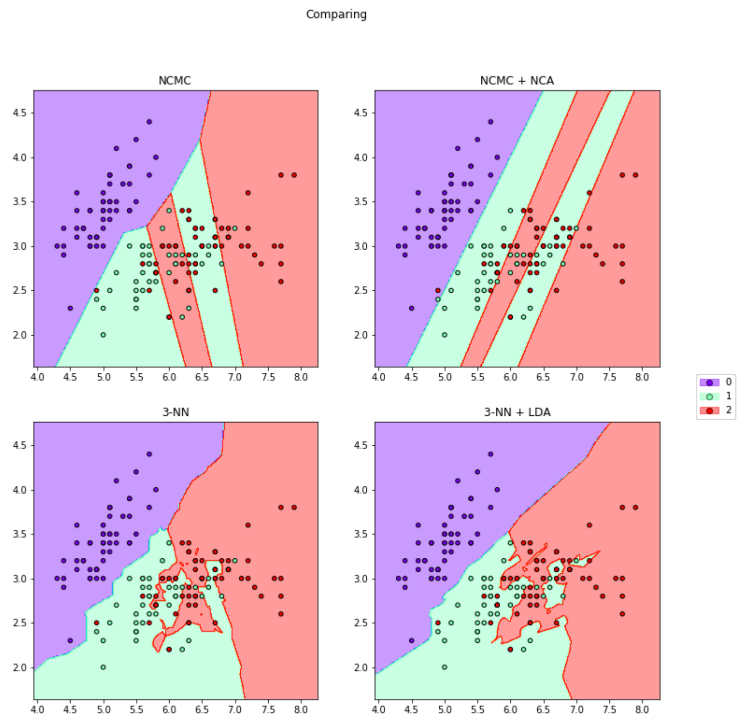

Figure 15: Plotting distance based classifiers with pyDML.

When in experiment 1 we talk about "capable of learning at maximum dimension" we are excluding those dimensionality reduction algorithms that only learn a change of axes, as is the case of PCA and ANMM, which at maximum dimension learn a transformation whose associated distance is still the euclidean. LDA is kept, assuming that it will always take the maximum dimension that it is able to, according to the number of classes of the problem. The algorithms oriented to centroid-based classifiers are also excluded from experiment 1, together with those based on kernels, which will be analyzed in the experiments 2 and 3, respectively.

The stated experiments indicate that the magnitude with which we will measure the performance of the algorithms is the result of the k-neighbors classification, except in the case of the algorithms based on centroids, which will use their corresponding classifier. These classifiers will be evaluated by a 10-fold cross validation. The results obtained from the predictions on the training set will also be included, in order to evaluate possible overfitting. The algorithms will be executed using their default parameters, which can be found in the `pyDML` documentation. These default parameters have been set with standard values. The following exceptions to the default parameters have been made:

- The LSI algorithm will have the parameter `supervised = True`, as it will be used for supervised learning.
- In the dimensionality reduction experiment (4), the algorithms will have the dimension number parameter set according to the dimension being evaluated.
- LMNN and KLMNN will have their parameter `k` equal to the number of neighbors being considered in the nearest neighbors classification.
- LMNN will be executed with stochastic gradient descent, instead of semidefinite programming, in dimensionality reduction experiments, thus learning a linear transformation instead of a metric.
- ANMM and KANMM will have their parameters `n_friends` and `n_enemies` equal to the number of neighbors being considered in the nearest neighbors classification.
- DMLMJ and KDMLMJ will have their parameter `n_neighbors` equal to the number of neighbors being considered in the nearest neighbors classification.
- NCMC will have its parameter `centroids_num` equal to the parameter `centroids_num` being considered in its corresponding classifier, `NCMC_Classifier`.

As for the datasets used in the experiments, up to 34 datasets have been collected, all of them available in KEEL[15]. All these datasets are numeric and without missing values, being oriented to standard classification problems. In addition, although some of the distance metric learning algorithms scale well with the number of samples, others cannot deal with datasets that are too large, so it was decided to select, for sets with a high number of samples, a subset of size that all algorithms can deal with, keeping the class distribution the same.

15. KEEL, *knowledge extraction based on evolutionary learning* (Triguero et al., 2017): `http://www.keel.es/`.

Thus, the characteristics of the datasets are described in Table 2. All datasets have been *min-max* normalized to the interval $[0, 1]$, feature to feature, prior to the execution of the experiments.

Dataset	Number of samples	Number of features	Number of classes
appendicitis	106	7	2
balance	625	4	3
bupa	345	6	2
cleveland	297	13	5
glass	214	9	7
hepatitis	80	19	2
ionosphere	351	33	2
iris	150	4	3
monk-2	432	6	2
newthyroid	215	5	3
sonar	208	60	2
wine	176	13	3
movement_libras	360	90	15
pima	768	8	2
vehicle	846	18	4
vowel	990	13	11
wdbc	569	30	2
wisconsin	683	9	2
banana (20 %)	1,060	2	2
digits	1,797	64	10
letter (10 %)	2,010	16	26
magic (10 %)	1,903	10	2
optdigits	1,127	64	10
page-blocks (20 %)	1,089	10	4
phoneme (20 %)	1,081	5	2
ring (20 %)	1,480	20	2
satimage (20 %)	1,289	36	7
segment (20 %)	462	19	7
spambase (10 %)	460	57	2
texture (20 %)	1,100	40	11
thyroid (20 %)	1,440	21	3
titanic	2,201	3	2
twonorm (20 %)	1,481	20	2
winequality-red	1,599	11	11

Table 2: Datasets used in the experiments.

Finally, we describe the details of the experiments 1, 2, 3 and 4:

1. Algorithms will be evaluated with the classifiers 3-NN, 5-NN and 7-NN.

2. NCMML will be evaluated with the `Scikit-Learn` NCM clasifier, while NCMC will be evaluated with the associated `pyDML` classifier for two different values: 2 centroids per class and 3 centroids per class.

3. Algorithms will be evaluated with 3-NN classifier, using the following kernels: linear (`Linear`), grade-2 (`Poly-2`) and grade-3 (`Poly-3`) polynomials, gaussian (`RBF`) and

laplacian (`Laplacian`). For the comparison, the kernel version of PCA[16] will be also included. In this case, only the smallest datasets will be considered, so that they can be applicable to the algorithms that scale the worst with the dimension (recall that the kernel trick forces algorithms to work in dimensions of the order of the number of samples).

4. Algorithms will be evaluated with the classifiers 3-NN, 5-NN and 7-NN. The dimensions to use will be: $1, 2, 3, 5, 10, 20, 30, 40, 50$, the maximum dimension of the dataset, and the number of classes of the dataset minus 1. In this case, the following high-dimensionality datasets are selected: `sonar`, `movement_libras` and `spambase`. The algorithms to be evaluated in this expriment will be: PCA, LDA, ANMM, DMLMJ, LMNN and NCA.

6.2 Results

This section shows the results of the cross-validation for the different experiments. We will show in this text only the results of the 3-NN classifier, for those experiments that use nearest neighbors classifiers. The results obtained for the remaining k-NN used in the experiments are available on the pyDML-Stats[17] website, where all the results are stored and kept up to date with the different changes in our software. The scripts used to do the experiments can also be found in this website. To the results of the experiments 1, 2 and 3 we have added the average score obtained, and the average ranking. This ranking has been made by assigning integer values between 1 and 10 (adding half fractions in case of a tie) according to the position of the algorithms over each dataset, 1 being the best algorithm, and m the worst one, where m is the number of algorithms being compared in each experiment. The content of the different tables elaborated is described below.

- Table 3 shows the cross-validation results obtained for experiment 1, using the 3-NN score as evaluation measure. Some cells do not show results because the algorithm did not converge.
- Table 4 shows the results of experiment 2. The evaluation measures were the NCM and NCMC classifiers with 2 and 3 centroids per class. For each classifier, the euclidean distance (`Euclidean + CLF`) and the distance learning algorithm associated with the classifier (`NCMML / NCMC (2 ctrd) / NCMC (3 ctrd)`) have been evaluated.
- Table 5 shows the cross-validation results obtained on the training set for the kernel-based algorithms using the 3-NN classifier. Table 6 shows the corresponding results on the test set.
- Table 7 shows the cross-validation results for the experiment 4 in dataset `sonar`, using the classifier 3-NN. On the left are the results for the training set, and on the right, the results for the test set. Each row shows the results for the different dimensions

16. It is implemented in `Scikit-Learn`: `http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html`. Its theoretical details can be found in Schölkopf et al. (1998).
17. Source code: `https://github.com/jlsuarezdiaz/pyDML-Stats`. The current website is located at `https://jlsuarezdiaz.github.io/software/pyDML/stats/`

evaluated. Tables 8 and 9 show the corresponding dimensionality results over the datasets `movement_libras` and `sonar`, respectively.

	Euclidean		LDA		ITML		DMLMJ		NCA		LMNN		LSI		DML-eig		MCML		LDML	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
appendicitis	.8428	.8339	.8428	.8522	.8533	.8604	.8490	.8256	**.8700**	.8504	.8407	.8422	.8659	**.8630**	.8585	.8622	.8502	.8513	.8669	.8422
balance	.8049	.8082	.8885	.8992	.8986	.8943	.8286	.8191	**.9592**	**.9584**	.8202	.8175	.9182	.9280	.8947	.8945	.8816	.8737	.8874	.8895
bupa	.6231	.6546	.6338	.6465	.6466	.6281	.6660	**.6776**	**.6943**	.5994	.6099	.6342	.6363	.6284	.5993	.6120	.5716	.5742	.5826	.5854
cleveland	.5570	.5468	.5694	.5502	.5488	.5523	.5626	.5636	**.6804**	.5436	.5780	.5803	.5518	.5722	.5896	.5829	.5978	.5785	.5784	**.5972**
glass	.6759	.7015	.6235	.6231	.6453	.6549	**.7092**	.7041	.7065	.6917	.6780	**.7067**	.6495	.6235	.6407	.6263	.6319	.5850	.6242	.6063
hepatitis	.8236	.8325	.9402	.8609	.9002	.8815	.8821	.8894	**.9569**	.8325	.9514	.8418	.9139	.9130	.9125	**.9176**	.9250	.8829	.9458	.8547
ionosphere	.8569	.8550	.8834	.8394	.8771	.8862	.8752	.8605	**.9534**	**.9084**	.9281	.8859	.8898	.8768	.8904	.8741	.9053	.8630	.8907	.8512
iris	.9533	.9533	.9681	.9533	.9703	.9733	.9585	.9666	.9755	.9666	.9481	.9400	.9703	**.9800**	.9585	.9600	.9688	.9466	**.9807**	.9600
monk-2	.9578	.9655	.9451	.9561	.9223	.9352	.9709	.9724	**1.000**	**1.000**	.9812	.9816	**1.000**	**1.000**	.9878	.9909	.9665	.9676	.9382	.9495
newthyroid	.9421	.9538	.9596	.9586	.9452	.9398	.9436	.9448	**.9700**	.9722	.9658	**.9725**	.9591	.9634	.9602	.9629	.9565	.9582	.9509	.9675
sonar	.8317	.8370	.9011	.7782	.8435	.8120	.9097	.8361	.9823	.8703	**.9941**	**.8742**	.8531	.8506	.8547	.7975	.8755	.8563	.8766	.7886
wine	.9606	.9606	.9968	**.9888**	.9900	.9773	.9812	.9662	.9956	.9882	.9956	.9832	.9837	.9662	.9975	.9767	**.9975**	.9832	.9956	**.9888**
movement_libras	.7972	.8139	**.8685**	.6642	.8038	.7992	.8460	**.8649**	.8516	.8319	.8065	.8020	.7351	.7440	.7970	.7872	.8063	.8073	.7256	.7360
pima	.7372	.7396	.7259	**.7525**	.7148	.7149	.7366	.7422	**.7841**	.7370	.7290	.7278	.7206	.7395	.7174	.7266	.7173	.7239	.7285	.7240
vehicle	.7077	.7125	.7698	**.7623**	.7625	.7516	.7643	.7551	**.8186**	.7550	.6855	.6757	.6590	.6666	.6506	.6501	.7398	.7369	.7186	.7170
vowel	.9699	.9787	.9680	.9777	.9423	.9535	.9751	**.9808**	**.9799**	**.9808**	.9693	.9777	.9436	.9474	.6719	.6757	.8558	.8737	.8885	.9090
wdbc	.9679	**.9716**	.9732	.9664	.9714	.9664	.9669	.9648	**.9751**	.9700	.9638	.9630	.9705	.9682	.9546	.9507	.9714	.9648	.9476	.9438
wisconsin	.9694	.9678	.9663	.9677	.9609	.9590	.9695	.9678	**.9723**	.9648	.9692	.9663	.9684	**.9722**	.9673	.9707	.9585	.9546	.9650	.9663
banana	.8543	.8555	.6504	.6469	.8536	.8556	.8550	.8565	.8553	**.8583**	**.8574**	.8583	.8535	.8517	.6718	.6878	.6282	.6102	.6268	.6319
digits	.9878	.9866	.9769	.9683	.9798	.9728	.9869	.9834	.9980	**.9894**	**.9993**	.9860	.9264	.9102	.8269	.8168	.9734	.9688	.9797	.9816
letter	.7174	.7208	.7955	.7967	.7161	.7195	.8163	.8204	**.8565**	**.8610**	.7048	.7162	.5396	.5496	.3191	.3214	.7600	.7534	.6217	.6372
magic	.8070	.8050	.7436	.7361	.8069	.8061	.8161	.8071	**.8396**	**.8145**	.7979	.7945	.7946	.7924	.7508	.7525	.7738	.7766	.7077	.6951
optdigits	.9756	.9777	.9671	.9512	.9731	.9669	.9770	.9761	.9956	.9759	**.9986**	**.9840**	.9398	.9306	.8164	.8022	.9761	.9591	.9596	.9591
page-blocks	.9495	.9495	**.9697**	**.9679**	.9614	.9614	.9515	.9504	.9637	.9577	.9459	.9439	-	-	.9515	.9523	.9613	.9642	.9438	.9404
phoneme	.7957	.7992	.7321	.7243	.7853	.7770	.7960	**.8002**	**.8044**	.7936	.7928	.7946	.7642	.7668	.7361	.7483	.7654	.7632	.7320	.7112
ring	.6410	.6432	.7289	.7101	.7290	.7352	.6440	.6453	**.9267**	**.8459**	.6750	.6615	.8331	.8162	.7308	.7223	.8315	.8223	.5634	.5648
satimage	.8585	.8564	.8541	.8387	.8495	.8341	.8670	**.8643**	**.8764**	.8511	.8565	.8558	.8490	.8465	.8153	.8130	.8246	.8171	.5427	.5501
segment	.8970	.9020	.9353	**.9370**	.9357	.9265	.9071	.9081	**.9451**	.9187	.9076	.8928	.8898	.8853	.9095	.9068	.9367	.9319	.8818	.8710
spambase	.8500	.8654	.9215	.8871	.8801	.8766	.8635	.8525	**.9391**	**.9154**	.9215	.9070	.9210	.9111	.9077	.9046	.9176	.9047	.9229	.8982
texture	.9560	.9618	**.9983**	**.9981**	.9801	.9754	.9864	.9854	.9843	.9800	.9180	.9218	.9333	.9400	.8979	.9009	.9740	.9745	.8658	.8718
thyroid	.9313	.9319	.9375	.9450	.9397	.9402	.9355	.9361	.9459	.9395	.9320	.9319	.9357	.9354	.9458	.9485	.9377	.9320	**.9587**	**.9583**
titanic	.7607	.7583	**.7727**	**.7804**	.7682	.7609	.7612	.7587	.5709	.6764	.6018	.6964	-	-	.7107	.7331	.7150	.7253	.7108	.7341
twonorm	.9609	.9595	.9778	.9750	.9685	.9669	.9612	.9561	**.9817**	.9790	.9778	.9756	.9776	.9770	.9782	**.9810**	.9708	.9730	.9789	.9804
winequality-red	.5808	**.5865**	.5657	.5733	.5754	.5828	.5828	.5860	**.6022**	.5766	.5647	.5772	.5656	.5809	.5281	.5292	.5675	.5611	.5376	.5471
AVG RANKING	6.558	5.661	5.147	5.426	5.808	5.382	4.926	4.544	**1.705**	**3.661**	5.220	5.279	6.411	5.382	6.691	6.220	5.735	6.279	6.794	7.161
AVG SCORE	.8383	.8425	.8515	.8363	.8500	.8470	.8560	.8526	**.8886**	**.8634**	.8490	.8432	.8410	.8405	.8059	.8041	.8438	.8359	.8125	.8062

Table 3: Results of cross-validation with 3-NN.

	Euclidean + NCM		NCMML		Euclidean + NCM (2 ctrd)		NCMC (2 ctrd)		Euclidean + NCM (3 ctrd)		NCMC (3 ctrd)	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
appendicitis	.8365	**.8640**	**.8512**	.8440	.6479	.6031	.6248	.6246	.8102	.7800	.7557	.7266
balance	.7496	.7475	.6865	.6864	.7004	.6721	.8280	**.8225**	.7160	.6654	**.8321**	.8222
bupa	.5996	.6004	**.6628**	**.6407**	.6270	.6058	.6247	.6260	.6589	.5903	.6409	.5826
cleveland	.5742	.5377	.6296	**.5516**	.5727	.5050	.6280	.5093	.6203	.4871	**.6435**	.5135
glass	.5218	.4862	.6221	.5290	.6479	.5849	.5877	.5372	.6427	.5208	**.7461**	**.6421**
hepatitis	.8500	.8293	.9832	**.8688**	.8792	.8436	.9513	.8373	.8986	.7946	**.9833**	.8672
ionosphere	.7445	.7378	.9313	.8797	.9186	**.8889**	.9018	.8764	.9062	.8750	**.9477**	.8886
iris	.9318	.9133	**.9814**	.9600	.9696	**.9666**	.9711	.9600	.9696	.9466	.9800	.9600
monk-2	.8078	.8104	.7950	.7923	.8071	.8082	.8112	.8038	.8127	.7895	**.8457**	**.8237**
newthyroid	.9359	.9352	**.9798**	.9634	.9498	.9448	.9741	**.9725**	.9695	.9681	.9757	.9629
sonar	.7265	.7017	.9257	.7641	.7120	.6929	.9219	.7839	.8360	.7437	**.9412**	**.7982**
wine	.9687	.9495	**1.000**	.9663	.9825	.9659	.9918	**.9826**	.9762	.9432	.9912	.9704
movement_libras	.6358	.5946	.8176	.7575	.7777	.6764	.8803	.7852	.8717	.7749	**.9420**	**.8373**
pima	.7337	.7279	**.7717**	**.7604**	.7565	.7382	.6720	.6641	.7521	.7461	.7322	.7253
vehicle	.4545	.4491	**.7998**	**.7797**	.6066	.5824	.7429	.7219	.6537	.6179	.7558	.7244
vowel	.5367	.5070	.6689	.6383	.6087	.5717	.7272	.6868	.5732	.5333	**.7690**	**.7292**
wdbc	.9388	.9367	.9794	.9649	.9548	.9385	.9796	**.9736**	.9703	.9665	**.9810**	.9701
wisconsin	.9648	.9648	**.9681**	**.9662**	.9586	.9502	.9655	.9603	.9515	.9486	.9541	.9487
banana	.5769	.5737	.5571	.5558	.6417	.6359	.5846	.5859	.7516	.7573	**.7828**	**.7766**
digits	.9066	.8981	.8047	.8083	.9521	.9415	.8664	.8586	**.9723**	**.9644**	**.8893**	.8554
letter	.5707	.5341	.7114	.6868	.5895	.5282	.7251	.6902	.6601	.5714	**.7825**	**.7301**
magic	.7712	.7693	.7790	.7751	.7786	.7745	**.7947**	**.7861**	.7600	.7509	.7820	.7751
optdigits	.9173	.9104	.8018	.7978	.9479	.9352	.8542	.8185	**.9675**	**.9609**	.8923	.8641
page-blocks	.8133	.8165	**.9636**	**.9576**	.8219	.8221	.9090	.9071	.8680	.8696	.8966	.8953
phoneme	.7417	.7399	.7587	.7538	**.7683**	**.7667**	.7084	.7159	.7207	.7149	.7091	.7048
ring	.7780	.7723	.7819	**.7784**	.8002	.7601	.7197	.7012	**.8160**	.7750	.6844	.6636
satimage	.7868	.7844	**.8478**	**.8255**	.8052	.7921	.8342	.8123	.8244	.7844	.8362	.8007
segment	.8460	.8367	**.9437**	**.9037**	.8596	.8452	.9203	.8976	.8581	.8030	.9242	.8874
spambase	.8874	.8827	**.9584**	.9154	.8816	.8763	.9400	**.9241**	.8985	.8893	.9335	.9112
texture	.7445	.7372	**.9912**	**.9781**	.8586	.8500	.9694	.9581	.9079	.8900	.9759	.9654
thyroid	.4532	.4394	**.8163**	**.8082**	.5724	.5558	.6875	.6922	.5959	.5630	.7469	.7358
titanic	.7540	.7459	**.7825**	**.7854**	.6746	.6516	.5624	.6550	.5630	.6824	.7279	.7350
twonorm	.9807	**.9824**	.9854	.9797	.9799	.9723	.9847	.9790	.9787	.9743	**.9855**	.9777
winequality-red	.3519	.3371	**.4535**	**.4359**	.4067	.3838	.4031	.3870	.3940	.3582	.4024	.3738
AVG RANKING	4.941	4.441	**2.382**	**2.500**	4.117	4.000	3.411	2.911	3.764	4.235	**2.382**	2.911
AVG SCORE	.7468	.7369	.8233	.7958	.7770	.7538	.8014	.7793	.7978	.7647	**.8344**	**.7984**

Table 4: Results of the experiments with NCM and NCMC.

	EUC	KPCA					KDA					KANMM					KDMLMJ					KLMNN				
		Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.
appendicitis	.8428	.8428	.8407	.8355	.8428	.8438	.8481	.8491	.8502	.8679	.8564	.8543	.8543	.8523	.8763	.8889	.8491	.8397	.8407	.8575	.8805	.8355	.8397	.8355	.8523	**1.000**
balance	.8049	.8217	.7770	.7923	.8136	.8288	.4547	.6756	.6279	.7497	.8838	.7764	.7980	.8224	.8462	.8327	.8298	.9509	.9696	.9374	.9491	.8202	.8426	.9079	.8865	**.9852**
bupa	.6231	.6231	.6235	.6222	.6231	.6560	.5623	.5677	.5758	.5526	.5665	.5903	.5890	.5832	.5639	.5735	.6586	.6254	.6457	.6247	.7075	.6437	.6547	.6627	.6721	**.9987**
cleveland	.5570	.5570	.5537	.5552	.5570	.5417	.5301	.5424	.5480	.5585	.5394	.5664	.5705	.5698	.5559	.5435	.5615	.5514	.5544	.5473	.5499	.5536	.5862	.6389	.6753	**.9943**
glass	.6759	.6759	.6801	.6801	.6759	.6931	.6376	.6723	.6459	.6474	.6661	.6884	.6853	.6874	.6724	.6848	.6977	.6957	.7024	.6702	.7342	.6790	.7200	.7289	.7372	**.9890**
hepatitis	.8236	.8236	.8083	.8111	.8236	.8403	.8318	.8416	.8471	.8181	.8111	.8695	.8709	.8653	.8363	.8306	.8834	.8708	.9179	.8708	.9208	.9777	**1.000**	**1.000**	**1.000**	**1.000**
ionosphere	.8569	.8569	.8518	.8493	.8569	.8882	.6986	.6749	.6860	.7641	.7429	.8512	.8518	.8502	.9259	.9449	.8752	.8714	.8774	.8819	.9398	.8695	.9670	.9692	.9838	**1.000**
iris	.9533	.9533	.9540	.9548	.9533	.9518	.8555	.9429	.9488	.9562	.9362	.9459	.9503	.9518	.9451	.9466	.9592	.9503	.9555	.9474	.9592	.9651	.9681	.9659	.9681	**1.000**
monk-2	.9578	.9503	.9603	.9580	.9480	.9511	.7047	.7119	.6995	.8320	.8459	.6877	.6365	.6252	.8996	.9964	.9709	.9763	.9863	.9588	.9938	.9789	.9873	.9920	.9989	**1.000**
newthyroid	.9421	.9421	.9395	.9385	.9421	.9488	.9581	.9576	.9571	.9638	.9612	.9390	.9400	.9410	.9617	.9643	.9441	.9478	.9421	.9503	.9612	.9690	.9700	.9731	.9741	**1.000**
sonar	.8317	.8317	.8365	.8370	.8317	.8392	.6261	.6287	.6410	.6057	.6832	.7499	.7516	.7521	.8263	.8450	.9076	.8696	.8637	.8691	.9332	.9706	**1.000**	**1.000**	**1.000**	**1.000**
wine	.9606	.9606	.9625	.9606	.9606	.9669	.9269	.9169	.9200	.9644	.9694	.9325	.9332	.9325	.9844	.9825	.9825	.9793	.9831	.9819	.9956	.9993	**1.000**	**1.000**	**1.000**	**1.000**
movement_libras	.7972	.7972	.7866	.7805	.7972	.7775	.3322	.4953	.4980	.7419	.7544	.4684	.4684	.4689	.7195	.7376	.8590	.8118	.8272	.8124	.8450	.7945	.8776	.8924	.9246	**1.000**
pima	.7372	.7372	.7361	.7377	.7372	.7181	.6820	.6753	.6653	.6535	.6650	.7170	.7144	.7141	.7164	.7076	.7359	.7381	.7460	.7455	.7628	.7378	.7453	.7397	.7505	**.9968**
banana	.8543	.8546	.8549	.8562	.8545	.8536	.6730	.6622	.6672	.7037	.6493	.8593	.8603	.8624	.7854	.8167	.8554	.8545	.8551	.8423	.7643	.8546	.8546	.8547	.8540	**.9801**
optdigits	.9756	.9756	.9761	.9753	.9756	.9664	.9186	.9190	.9198	.9434	.9386	.9379	.9387	.9394	.9558	.9543	.9767	.9790	.9813	.9791	.9857	.9905	.9999	.9997	.9998	**1.000**
phoneme	.7957	.7957	.7940	.7935	.7957	.7962	.6960	.7019	.6929	.7112	.7271	.7727	.7726	.7732	.7735	.7880	.7959	.7940	.7945	.7898	.7858	.8021	.7963	.7850	.7994	**.9829**
satimage	.8585	.8585	.8583	.8594	.8585	.8641	.8104	.8114	.8097	.8422	.8452	.8188	.8186	.8190	.8535	.8601	.8707	.8623	.8622	.8590	.8680	.8623	.8682	.8682	.8803	**.9935**
segment	.8970	.8970	.8972	.8968	.8970	.9004	.8360	.8278	.8276	.8187	.8737	.8309	.8300	.8300	.8323	.8723	.9049	.8965	.8994	.8879	.9343	.9244	.9383	.9405	.9415	**.9985**
spambase	.8500	.8500	.8500	.8504	.8500	.8328	.8490	.8323	.8313	.7263	.7500	.8777	.8775	.8777	.8635	.8988	.8642	.8850	.8828	.8830	.9014	.9369	.9495	.9459	.9497	**.9990**
twonorm	.9609	.9609	.9636	.9648	.9609	.9531	.9765	.9758	.9758	.9765	.9756	.9798	.9800	.9804	.9763	.9751	.9616	.9642	.9624	.9672	.9810	.9773	.9847	.9800	.9861	**.9915**
AVG RANKING	15.45	15.26	15.52	15.95	15.47	14.14	22.19	21.64	21.50	18.95	19.47	17.14	16.61	16.30	15.52	13.66	9.690	11.71	9.880	12.33	7.023	9.476	5.380	5.833	3.619	**1.214**
AVG SCORE	.8360	.8365	.8336	.8338	.8360	.8387	.7337	.7563	.7541	.7808	.7924	.7959	.7949	.7952	.8271	.8402	.8545	.8530	.8595	.8506	.8740	.8639	.8833	.8895	.8969	**.9957**

Table 5: Results of kernel experiments on the training set.

	EUC	KPCA					KDA					KANMM					KDMLMJ					KLMNN				
		Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.	Lin.	Poly-2	Poly-3	RBF	Lapl.
appendicitis	.8339	.8339	.8339	.8248	.8339	.8546	.8613	.8813	.8813	.8713	.8622	.8646	.8446	.8446	.8813	**.8904**	.8339	.8248	.8339	.8339	.8248	.8322	.8248	.8331	.8057	.8422
balance	.8082	.8383	.7823	.8032	.8428	.8336	.4597	.6699	.6150	.7407	.8899	.7738	.8058	.8382	.8559	.8320	.8144	.9582	**.9711**	.9374	.9519	.8222	.8367	.9118	.8736	.8464
bupa	.6546	.6546	.6661	.6662	.6546	.6862	.5620	.5191	.5566	.5244	.5389	.5963	.6022	.5908	.6199	.5794	.6777	.6375	.6371	.6310	**.6924**	.6574	.6402	.6516	.6632	.6467
cleveland	.5468	.5468	.5432	.5535	.5468	.5528	.5541	.5449	.5485	**.5860**	.5474	.5689	.5653	.5685	.5657	.5623	.5605	.5682	.5774	.5674	.5688	.5736	.5744	.5628	.5184	.5325
glass	.7015	.7015	.7102	.7102	.7015	.7117	.5971	.6708	.6543	.6582	.6630	.6777	.6677	.6677	.7026	.6890	.7020	.6910	.6910	.6715	**.7351**	.6907	.6911	.6827	.6845	.7334
hepatitis	.8325	.8325	.8343	.8343	.8325	.8436	.8123	.8140	.8265	.8325	.8436	.8732	.8732	.8732	.8575	.8575	**.8894**	.8672	.8547	.8547	.8644	.8408	.8672	.8404	.8672	.8845
ionosphere	.8550	.8550	.8520	.8491	.8550	.8885	.7119	.6695	.6977	.7638	.7598	.8517	.8489	.8461	.9258	**.9457**	.8634	.8607	.8606	.8666	.9316	.8463	.9081	.8910	.8969	.9396
iris	.9533	.9533	.9533	.9533	.9533	.9533	.8800	.9533	.9533	**.9800**	.9333	.9600	.9600	.9533	.9533	.9466	.9600	.9600	.9533	.9533	.9533	.9533	.9533	.9600	.9466	.9266
monk-2	.9655	.9539	.9677	.9655	.9634	.9585	.7294	.7203	.6880	.8176	.8385	.6877	.6529	.6435	.9213	.9930	.9724	.9699	.9862	.9654	.9908	.9817	.9863	.9817	.9863	**1.000**
newthyroid	.9538	.9538	.9448	.9448	.9538	.9493	.9538	.9538	.9538	.9586	.9491	.9396	.9396	.9396	.9580	.9627	.9493	.9493	.9448	.9491	.9489	.9584	.9632	**.9679**	.9632	.9625
sonar	.8370	.8370	.8515	.8515	.8370	.8560	.5812	.6540	.6431	.5948	.6872	.7451	.7451	.7451	.8225	.8267	.8461	.8556	.8560	.8651	**.8753**	.7653	.8701	.8408	.8704	.8316
wine	.9606	.9606	.9606	.9606	.9606	.9613	.9214	.9047	.9158	.9367	.9603	.9262	.9318	.9318	.9888	.9835	.9606	.9780	.9724	.9777	.9780	**.9888**	.9830	.9666	**.9888**	.9777
movement_libras	.8139	.8139	.8070	.7948	.8139	.8059	.3715	.5209	.5187	.7631	.7600	.4969	.5006	.4982	.7615	.7566	**.8718**	.8251	.8406	.8219	.8234	.8106	.8315	.8093	.8375	.7989
pima	.7396	.7396	.7370	.7318	.7396	.7175	.6967	.6850	.7150	.6810	.6784	.7238	.7251	.7278	.7134	.7109	.7383	.7396	**.7513**	.7487	.7408	.7462	.7461	.7370	.7370	.7005
banana	.8555	.8555	.8546	.8546	.8555	.8574	.6688	.6642	.6934	.7030	.6169	.8583	.8592	**.8611**	.7810	.8177	.8565	.8546	.8536	.8425	.7622	.8536	.8508	.8565	.8565	.8414
optdigits	.9777	.9777	.9795	.9777	.9777	.9706	.9146	.9155	.9164	.9419	.9350	.9359	.9359	.9377	.9565	.9529	.9752	**.9804**	.9787	.9777	.9804	.9607	.9760	.9697	.9671	.9572
phoneme	.7992	.7992	.7964	.7973	.7992	.8002	.6847	.7068	.7067	.7132	.7270	.7826	.7845	.7854	.7724	.7880	.8030	.8029	.7964	.7918	.7845	**.8047**	.8001	.7854	.7991	.7835
satimage	.8564	.8564	.8564	.8580	.8564	.8612	.8053	.8138	.8122	.8364	.8496	.8193	.8185	.8170	.8535	.8589	**.8689**	.8589	.8580	.8527	.8559	.8473	.8565	.8487	.8425	.8558
segment	.9020	.9020	.9020	.9000	.9020	.9000	.8404	.8363	.8323	.8112	.8625	.8346	.8346	.8326	.8227	.8687	.9098	.8768	.8840	.8738	.9183	.9153	.9241	.9224	.9285	.9517
spambase	.8654	.8654	.8676	.8676	.8632	.8110	.8372	.8348	.8282	.7151	.7303	.8807	.8807	.8807	.8567	.8849	.8546	.8740	.8784	.8826	.8827	.8914	.9027	.8981	**.9049**	.8999
twonorm	.9595	.9595	.9649	.9642	.9595	.9567	.9804	.9777	.9770	.9797	.9730	**.9810**	.9797	.9790	.9743	.9682	.9561	.9635	.9574	.9675	.9702	.9689	.9702	.9682	.9655	.9635
AVG RANKING	12.97	12.83	13.21	13.57	12.76	11.73	21.42	20.73	20.64	17.92	19.21	15.57	15.95	16.16	13.40	12.23	9.738	9.642	10.21	11.61	8.404	11.02	**7.880**	10.38	10.02	11.69
AVG SCORE	.8415	.8424	.8412	.8411	.8430	.8443	.7345	.7577	.7588	.7814	.7908	.7990	.7979	.7982	.8354	.8417	.8507	.8522	.8541	.8492	**.8588**	.8433	.8551	.8517	.8525	.8512

Table 6: Results of kernel experiments on the test set.

	PCA	LDA	ANMM	DMLMJ	NCA	LMNN
1	.5016	.9011	.6965	.7826	.9214	.7237
2	.5891	-	.7670	.8050	.9807	.8782
3	.7729	-	.8359	.8333	.9770	.9513
5	.8215	-	.8904	.9033	.9759	.9914
10	.8600	-	.8958	.9652	.9764	.9994
20	.8541	-	.8872	.9583	.9668	**1.000**
30	.8456	-	.8627	.9508	.9706	**1.000**
40	.8365	-	.8424	.9460	.9839	**1.000**
50	.8312	-	.8370	.9263	.9850	**1.000**
Max. Dimension	.8317	-	.8317	.9097	.9823	**1.000**
N. Classes - 1	.5016	.9011	.6965	.7826	.9214	.7237

	PCA	LDA	ANMM	DMLMJ	NCA	LMNN
1	.5619	.7782	.6770	.7256	.8073	.6640
2	.6293	-	.7541	.7113	.8077	.7593
3	.7641	-	.8395	.7741	.8265	.8077
5	.8075	-	.8263	.8182	.8220	.8408
10	.8699	-	.8751	.8651	.8270	.8654
20	.8601	-	.8749	**.8844**	.8699	.8703
30	.8610	-	.8649	.8749	.8653	.8754
40	.8465	-	.8610	.8697	.8792	.8613
50	.8565	-	.8515	.8654	.8558	.8706
Max. Dimension	.8370	-	.8370	.8361	.8703	.8706
N. Classes - 1	.5619	.7782	.6770	.7256	.8073	.6640

Table 7: Results of dimensionality reduction experiments on `sonar` with 3-NN (train - test)

	PCA	LDA	ANMM	DMLMJ	NCA	LMNN
1	.1938	.3339	.2414	.2720	.3360	.2547
2	.2813	.5362	.4597	.4720	.6638	.5416
3	.5232	.6143	.6435	.6684	.7195	.6900
5	.6873	.7211	.7473	.7918	.8188	.8156
10	.7831	.8661	.8053	.8857	.8383	.8485
20	.7978	-	.7972	.8705	.8442	.8490
30	.7981	-	.7978	.8652	.8438	.8514
40	.7972	-	.7975	.8594	.8469	.8526
50	.7972	-	.7972	.8538	.8431	.8498
Max. Dimension	.7972	-	.7972	.8460	.8516	.8490
N. Classes - 1	.7932	.8685	.8061	**.8901**	.8398	.8438

	PCA	LDA	ANMM	DMLMJ	NCA	LMNN
1	.1747	.3169	.2675	.2694	.2606	.2673
2	.2574	.4553	.4800	.4476	.6181	.5251
3	.5483	.4978	.6680	.6684	.6920	.6499
5	.7177	.5938	.7763	.7774	.7655	.8012
10	.8007	.7001	.8119	.8711	.8017	.8220
20	.8139	-	.8106	**.8829**	.8143	.8333
30	.8139	-	.8139	.8696	.8191	.8133
40	.8139	-	.8139	.8605	.8323	.8233
50	.8139	-	.8139	.8627	.8310	.8255
Max. Dimension	.8139	-	.8139	.8649	.8319	.8133
N. Classes - 1	.8185	.6642	.8137	.8811	.8274	.8211

Table 8: Results of dimensionality reduction experiments on `movement_libras` with 3-NN (train - test)

	PCA	LDA	ANMM	DMLMJ	NCA	LMNN
1	.8369	.9215	.8567	.6995	**.9420**	.9340
2	.8316	-	.8869	.7724	**.9420**	.9386
3	.8487	-	.8973	.8886	.9388	.9335
5	.8784	-	.9079	.9009	.9415	.9335
10	.8681	-	.9222	.9195	.9400	.9318
20	.8700	-	.9067	.9217	.9400	.9297
30	.8586	-	.8787	.8867	.9403	.9328
40	.8572	-	.8654	.8727	.9369	.9318
50	.8536	-	.8560	.8596	.9374	.9299
Max. Dimension	.8500	-	.8500	.8635	.9391	.9285
N. Classes - 1	.8369	.9215	.8567	.6995	**.9420**	.9340

	PCA	LDA	ANMM	DMLMJ	NCA	LMNN
1	.8106	.8871	.8587	.6588	.9044	.8872
2	.8261	-	.8850	.7173	**.9197**	.8958
3	.8543	-	.9090	.8807	.9152	.9068
5	.8782	-	.9049	.8700	.9111	.9069
10	.8826	-	.9198	.9044	.9153	.9113
20	.8695	-	.9048	.8937	.9155	.9005
30	.8502	-	.8675	.8851	.9154	.9027
40	.8547	-	.8567	.8611	.9111	.9005
50	.8655	-	.8633	.8569	.9133	.9070
Max. Dimension	.8654	-	.8654	.8525	.9154	.9092
N. Classes - 1	.8106	.8871	.8587	.6588	.9044	.8872

Table 9: Results of dimensionality reduction experiments on `spambase` with 3-NN (train - test)

6.3 Analysis of Results

On the results obtained in the first experiment, we can clearly see that NCA is the one that has obtained the best results. This is partly due to the fact that the algorithms have been evaluated with nearest neighbors classifiers, and NCA was specifically designed to improve this classifier. NCA got the first place in most of the validations over the training set, showing its ability to fit to the data, but it has also obtained clear victories in many of the datasets over the test set, thus also demonstrating a great capacity for generalization.

We can also see that DMLMJ and LMNN algorithms stand out, although at a considerable distance from NCA. These algorithms were also oriented to nearest neighbors classification, which justifies these good results. About LMNN we also conjecture that it has a slow convergence with the projected gradient method, and it could have achieved better results with a greater number of iterations. In fact, in the analysis of dimensionality reduction experiments we will see a much better performance of LMNN with the stochastic gradient descent method. LSI is another algorithm capable of obtaining very good results on certain datasets, but it is penalized by many others, where it is not able to optimize enough, even not converging in several datasets.

ITML and MCML are two algorithms that, despite getting the best results in a very small number of cases, they get decent results in most datasets, resulting in quite a stable performance. ITML does not learn too much from the characteristics of the training set, but is able to generalize what has been learned in a quite effective way, being possibly the algorithm that loses the least accuracy over the test set, with respect to the training set. On the other hand, MCML has more learning capacity, even showing a slight overfitting, as its results are worse than those of many algorithms on the test set.

Another algorithm in which we can see overfitting, perhaps more clearly, is LDA. This algorithm is capable of getting very good results on the training set, surpassing most of the algorithms, but it gets noticeably worse when evaluated on the test dataset. Recall that LDA is able to learn only a maximum dimension equal to the number of classes of the dataset minus one. This may be causing the loss of important information on many datasets by the projection it learns.

Finally, although DML-eig and LDML are able to get better results than euclidean distance on the training sets, on several datasets they have obtained quite low quality results. On many of the test datasets, they are surpassed by the euclidean distance.

If we analyze the results of the centroid-based classifiers, we can easily observe that in the vast majority of cases the classifier has worked much better after learning the distance with its associated learning algorithm, than using the euclidean distance. It can also be observed that the results are subject to great variability, depending on the number of centroids chosen. This shows that the choice of an adequate number of centroids, that adapts well to the disposition of the different classes, is fundamental to achieve a successful learning with these algorithms.

Focusing now on the kernel-based algorithms, it is interesting to note how KLMNN with laplacian kernel is able to adjust as much as possible to the data, getting a 100 % success rate on most of the datasets. This success rate is not transferred, in general, to the test data, showing that this algorithm overfits with laplacian kernel. We can also observe that the best results are distributed in a varied way among the different evaluated options. The choice of a suitable kernel that fits well with the disposition of the data is decisive for the performance of kernel-based algorithms.

To conclude our analysis, dimensionality experiments allow us to observe that the best results are not always obtained when considering the maximum dimension. This may be due to the fact that the algorithms are able to denoise the data, ensuring that the classifier used later does not overfit. We also see that we cannot reduce the dimension as much as we want, because at some point we start losing information, which happens in many cases with LDA, which is its great limitation. In general, we can observe that all algorithms improve their results by reducing dimensionality until a certain value, although the best results are provided by LMNN, DMLMJ and NCA. The results obtained by LMNN open the possibility of using this algorithm with stochastic gradient descent, instead of the semidefinite programming algorithm used in the first experiment, since the results it provides are quite good. Although these algorithms have obtained better results, the use of ANMM and LDA (as long as the dimension allows it) is important for the estimation of an adequate dimension, since they are much more efficient than the first ones. As for PCA, it gets the worst results in low dimensions, probably due to not considering the information of the labels.

To complete the verbal analysis carried out, we have developed a series of Bayesian statistical tests to assess the extent to which the performance of the different algorithms analyzed outperforms the others. To do this, we have elaborated several pairwise Bayesian sign tests (Benavoli et al., 2017). In these tests, we will consider the differences between the obtained scores of two algorithms, assuming that their prior distribution is a Dirichlet Process (Benavoli et al., 2014), defined by a prior strength $s = 1$ and a prior pseudo-observation $z_0 = 0$. After considering the score observations obtained for each dataset, we obtain a posterior distribution which gives us the probabilities that one algorithm outperforms the other. We also introduce a rope region, in which we consider the algorithms to have an equivalent performance. We have designated the rope region as the one where the score differences are in the interval $[-0.01, 0.01]$. In summary, from the posterior distribution we obtain three probabilities: the probability that the first algorithm outperforms the second, the probability that the second algorithm outperforms the first one, and the probability that both algorithms are equivalent. These probabilities can be visualized in a simplex plot for a sample of the posterior distribution, in which a greater tendency of the points towards one of the regions will represent a greater probability.

To do the Bayesian sign tests, we have used the R package `rNPBST` (Carrasco et al., 2017). In Figure 16 we pairwise compare some of the algorithms that seem to have a better performance in experiment 1 with 3-NN (NCA, DMLMJ and LMNN) with the results of the 3-NN classifier for euclidean distance. In the comparison between euclidean distance and NCA, we can clearly see that the points are concentrated close to the $[\mathrm{NCA}, \mathrm{rope}]$ segment. This shows us that euclidean distance is unlikely to outperform NCA, and there is also a high probability for NCA to outperform euclidean distance, since a big concentration of points is in the NCA region. We obtain similar conclusions for DMLMJ against euclidean distance, although in this case, despite the fact that euclidean distance is still unlikely to win, there is a greater concentration of points in the rope region. In the comparison between LMNN and euclidean distance, we see a more centered concentration of points, but slightly weighted towards the LMNN region. In the comparisons between the distance metric learning algorithms we observe the points weighted to the $[\mathrm{NCA}, \mathrm{rope}]$ segment, concluding the difficulty of outperforming NCA, and between DMLMJ and LMNN we can see a pretty level playing field, but slightly biased to the DMLMJ algorithm.

The outperforming of euclidean distance is even more clear in the results of experiment 2. For these algorithms, we can clearly observe the points concentrated in the region corresponding to the nearest centroid metric learning algorithm, as shown in Figure 17. We have elaborated more pairwise Bayesian sign tests for the rest of algorithms in experiment 1. The results of these tests are also available on the pyDML-Stats website[17].

7. Conclusions

In this tutorial we have studied the concept of distance metric learning, showing what it is, what its applications are, how to design its algorithms, and the theoretical foundations of this discipline. We have also studied some of the most popular algorithms in this field, also with their theoretical foundations, and explaining different resolution techniques.

In order to understand the theoretical foundations of distance metric learning and its algorithms, it has been necessary to delve into three different mathematical theories: convex analysis, matrix analysis and information theory. Convex analysis has made it possible to present many of the optimization problems studied in the algorithms, along with some methods for solving them. Matrix analysis has provided many useful tools to understand this discipline, from how to parameterize Mahalanobis distances, to the optimization with eigenvectors, going through the most basic algorithm of semidefinite programming. Finally, information theory has motivated several of the algorithms we have studied.

As for the developed software, we have integrated the distance metric learning algorithms in a library for the Python language, with additional functionalities such as classifiers, visualization or parameter estimation.

In addition, several experiments have been developed that have allowed to evaluate the performance of the algorithms in the softwate. The results of these experiments have allowed us to observe how algorithms such as LMNN, DMLMJ, and specially NCA can considerably improve the nearest neighbors classification, and how centroid-based distance learning algorithms also improve their corresponding classifiers. We have also seen the wide variety of possibilities offered by kernel-based algorithms, and the advantages that an appropiate reduction of the dimensionality of the datasets can offer.

8. Future Work

The work carried out in this paper has paved the way for future research, both in the study of distance metric learning and in the software developed. Some of these possibilites are:

- **Other approaches for the concept of distance.** Most of the current distance metric learning theory focus on Mahalanobis distances. However, some articles open a door to learning about other possible distances, such as local Mahalanobis distances, that lead to a multi-metric learning (see Weinberger and Saul, 2009). By developing new approaches, we will have a greater variety of distances to learn, and thus have a greater chance of success.

- **Kernelization of existing algorithms.** The kernelization of distance metric learning algorithms can be extended to other algorithms besides those presented. The search for a suitable parametrization and a representer theorem that allows the kernel trick to be applied is another possible task to carry out.

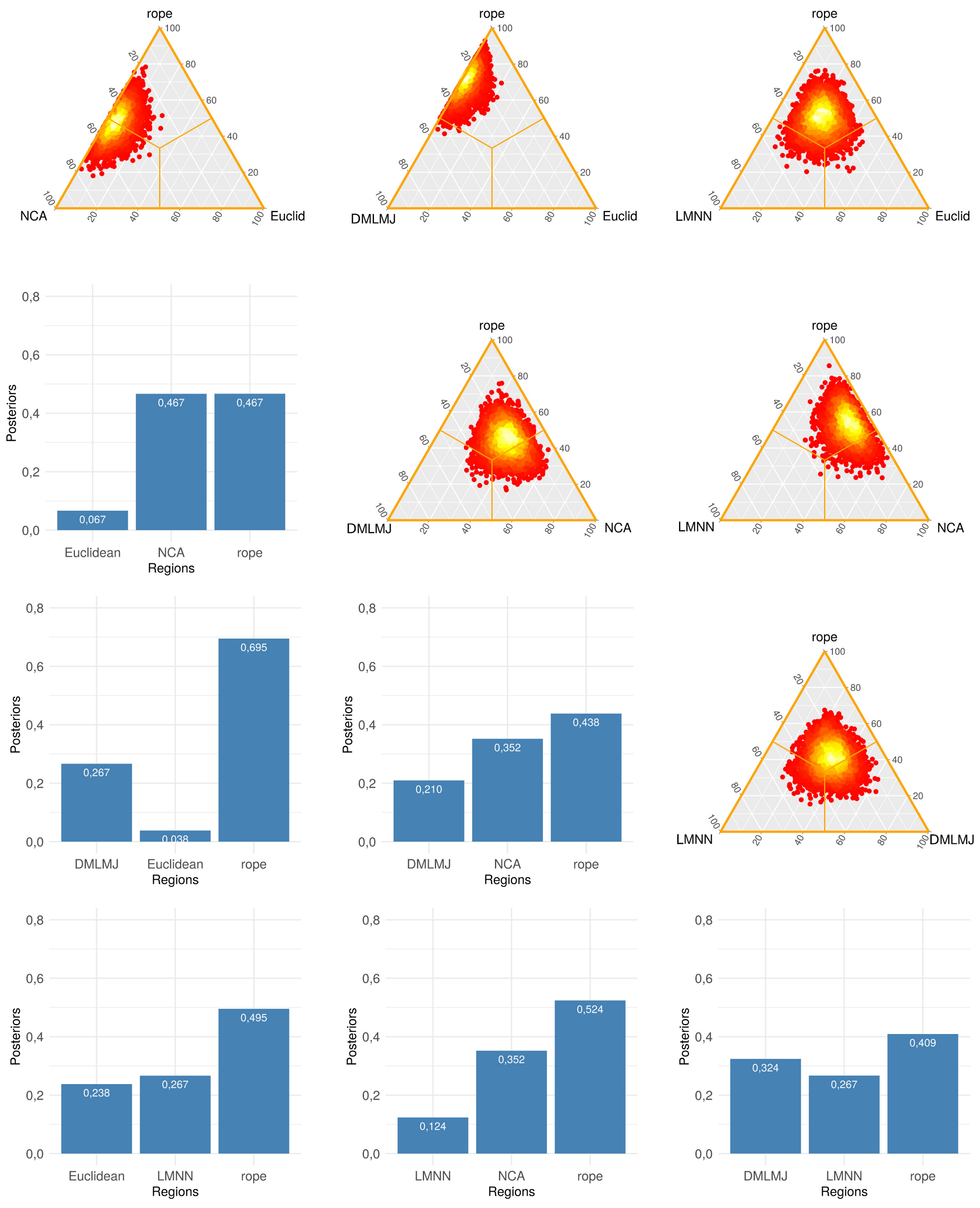

Figure 16: Bayesian sign results for NCA, DMLMJ, LMNN and euclidean distance with 3-NN.

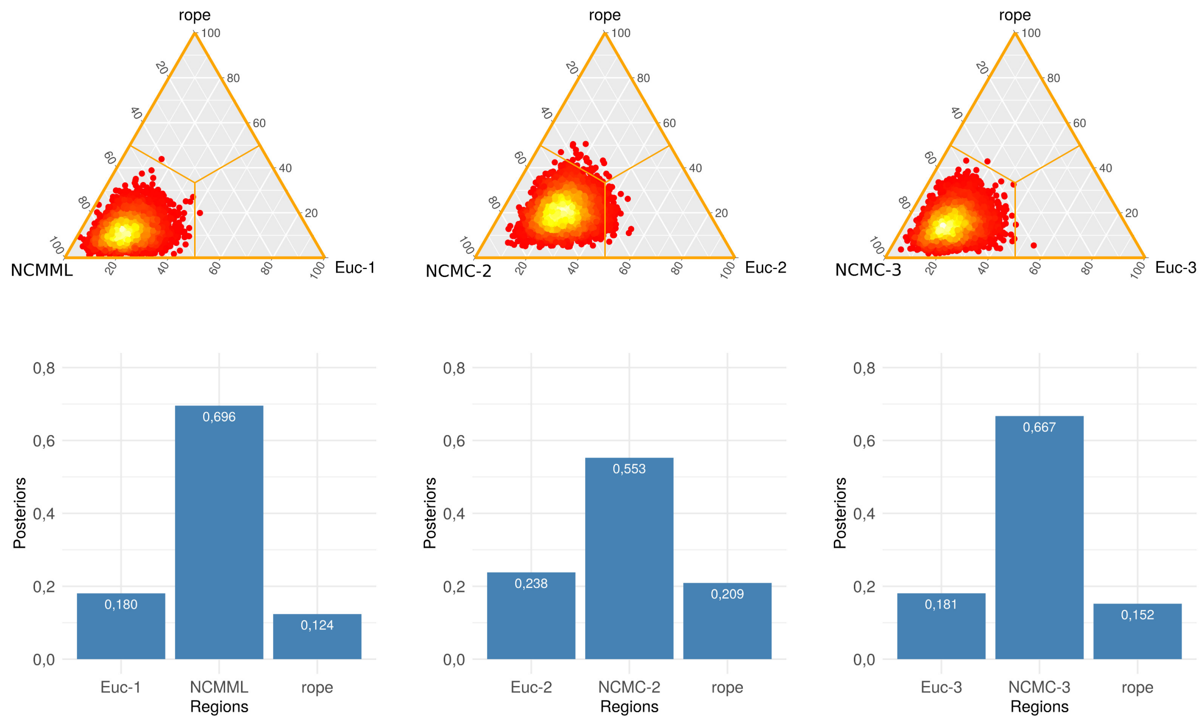

Figure 17: Bayesian sign test results for the comparison between scores of nearest centroid classifiers with their corresponding distance metric learning algorithm against the same classifier with euclidean distance. The results are shown for nearest class mean classifier (left), nearest class with 2 centroids (center) and nearest class with 3 centroids (right).

- **Other optimization mechanisms.** The algorithms studied optimize their objective functions by applying gradient descent method, regardless of whether the objective function is convex. The use of other optimization techniques, such as metaheuristics, can be useful to improve those algorithms that do not have convex objective functions. The evolutionary approach to distance metric learning has been explored recently in several problems (Kalintha et al., 2017; Ali et al., 2018).

- **High dimensionality datasets.** Distance metric learning is of great interest in many real problems in high dimensionality, such as face recognition, where it is very useful to be able to measure the similarity between different images (Moutafis et al., 2017). When we work with datasets of even greater dimensionality, the treatment of distances can become too expensive, since it would be necessary to store matrices of very large dimensions. In these situations it may be of interest to combine distance metric learning with feature selection techniques prepared for very high dimensional data (Tan et al., 2014).

- **Big Data solutions.** The problem of learning when the amount of data we have is huge and heterogeneous is one of the challenges of machine learning nowadays (Wu et al., 2014). In the case of the distance metric learning algorithms, although many of them, specially those based on gradient descent, are quite slow and do not scale well with the number of samples, they can be largely parallelized in both matrix computations and gradient descent batches. In this way, distance metric learning can be extended to handle Big Data by developing specialized algorithms and integrating them with frameworks such as Spark (Meng et al., 2016) and Cloud Computing architectures (Hashem et al., 2015).

- **Singular problems.** In this paper, we have focused on distance metric learning for usual problems, like standard classification and dimensionality reduction, and we have also mentioned its applications for clustering and semi-supervised learning. However, distance metric learning can be useful in a wide variety of learning tasks (Charte et al., 2019), and can be carried out either by designing new algorithms or by adapting known algorithms from standard problems to these tasks. In recent years, several distance metric learning proposals have been made in problems like regression (Nguyen et al., 2016), multi-dimensional classification (Ma and Chen, 2018), ordinal classification (Nguyen et al., 2018) and multi-output learning (Liu et al., 2018).

- **Hybridization with other learning techniques.** Over the years, some distance-based algorithms, or some of their ideas or foundations, have been combined with other algorithms in order to improve their learning capabilities in certain problems. For example, the concept of nearest-neighbors has been combined with classifiers such as Naive-Bayes, obtaining a Naive-Bayes classifier whose feature distributions are determined by the nearest neighbors of each class (Yang and Tian, 2012); with neural networks, to find the best neural network architecture (Wang et al., 2017); with random forests, by exploiting the relationship between voting points and potential nearest neighbors (Lin and Jeon, 2006); with deep learning, to provide interpretability and robustness to deep neural networks (Papernot and McDaniel, 2018); with ensemble methods, like bootstrap (Steele, 2009; Hamamoto et al., 1997); with support vector machines, training them locally in neighborhoods (Zhang et al., 2006); or with rule-learning algorithms, obtaining the so-called *nested generalized exemplar* algorithms (Wettschereck and Dietterich, 1995). The distances used in these combinations of algorithms can condition their performance, so designing appropiate distance learning algorithms for each of these tasks can be helpful for achieving good results. Following this topic, another option is to hybridize directly distance metric learning with other techniques, like ensemble learning (Mu et al., 2013).

Acknowledgments

Our work has been supported by the research project TIN2017-89517-P and by a research scholarship, given to the author Juan Luis Suárez by the University of Granada.

Appendix A. Positive Semidefinite Matrices and Decomposition Theorems.

In this appendix we will study several decomposition theorems involving positive semidefinite matrices, in order to prove Theorem 10. We will start with a characterization of positive semidefinite matrices by decomposition, which will also allow us to introduce the concept of square root. We will rely on several previous lemmas.

Lemma 30 *Let $A, B \in \mathcal{M}_d(\mathbb{R})$ be two commuting matrices, that is, $AB = BA$. Then, $Ap(B) = p(B)A$, where p denotes any polynomial over matrices (that is, a expression of the form $p(C) = a_0 I + a_1 C + a_2 C^2 + \cdots + a_n C^n$, with $a_0, \dots, a_n \in \mathbb{R}$).*

Proof Observe that

$$AB^n = (AB)B^{n-1} = B(AB)B^{n-2} = \cdots = B^{n-1}(AB) = B^n A,$$

and $Ap(B) = p(B)A$ is deduced by linearity. ■

Lemma 31 *Let $D \in S_d(\mathbb{R})_0^+$ be a diagonal matrix. Then, there is a polynomial over matrices p so that $p(D^2) = D$.*

Proof Suppose $D = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$, with $0 \le \lambda_1 \le \cdots \le \lambda_d$. Then, $D^2 = \operatorname{diag}(\lambda_1^2, \dots, \lambda_d^2)$. We take p as an interpolation polynomial over the points (λ_i^2, λ_i), for $i = 1, \dots, d$. If we evaluate it on D^2 we obtain

$$p(D^2) = p(\operatorname{diag}(\lambda_1^2, \dots, \lambda_d^2)) = \operatorname{diag}(p(\lambda_1^2), \dots, p(\lambda_d^2)) = \operatorname{diag}(\lambda_1, \dots, \lambda_d) = D.$$

■

Theorem 32 *Let $M \in \mathcal{M}_d(\mathbb{R})$. Then,*

1. *$M \in S_d(\mathbb{R})_0^+$ if, and only if, there is $L \in \mathcal{M}_d(\mathbb{R})$ so that $M = L^T L$.*
2. *If $M \in S_d(\mathbb{R})_0^+$, there is a single matrix $N \in S_d(\mathbb{R})_0^+$ with $N^2 = M$. In addition, $M \in S_d(\mathbb{R})^+ \iff N \in S_d(\mathbb{R})^+$.*

Proof , First we will see that $L^T L$ is a positive semidefinite, for any $L \in \mathcal{M}_d(\mathbb{R})$. Indeed, given $x \in \mathbb{R}^d$,

$$x^T L^T L x = (Lx)^T (Lx) = \|Lx\|_2^2 \ge 0.$$

We will prove the second implication of the first statement finding directly the matrix N of the second statement. Consider the spectral decomposition $M = UDU^T$, with $U \in O_d(\mathbb{R})$ and $D = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$, with $0 \le \lambda_1 \le \dots \lambda_d$ the eigenvalues of M. We define $D^{1/2} = \operatorname{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_d})$ and construct the matrix $N = UD^{1/2}U^T$. N is positive semidefinite, because its eigenvalues are those of $D^{1/2}$, which are all positive, and besides,

$$N^2 = UD^{1/2}U^T UD^{1/2}U^T = UD^{1/2}D^{1/2}U^T = UDU^T = M.$$

Furthermore, the strict positivity of the eigenvalues of M is equivalent to that of the eigenvalues of N, then $M \in \mathcal{M}_d(\mathbb{R})^+ \iff N \in \mathcal{M}_d(\mathbb{R})^+$. Let us finally see that N is unique.

Suppose that we have $N_1, N_2 \in S_d(\mathbb{R})_0^+$ with $N_1^2 = M = N_2^2$. Observe that N_1 and N_2 must have the same eigenvalues, since they are necessarily the positive square roots of the eigenvalues of M. Therefore, N_1 and N_2 are similar to a same diagonal matrix, that is, there are matrices $U, V \in O_d(\mathbb{R})$ with $N_1 = UDU^T$ and $N_2 = VDV^T$. From $N_1^2 = N_2^2$ we have

$$UD^2U^T = VD^2V^T \implies V^TUD^2 = D^2V^TU,$$

so for $W = V^TU \in O_d(\mathbb{R})$ we obtain that D^2 and W commute. Combining Lemmas 31 and 30, we obtain that D and W also commute. Therefore,

$$WD = DW \implies V^TUD = DV^TU \implies UDU^T = VDV^T \implies N_1 = N_2,$$

obtaining the uniqueness. ■

As we had anticipated, this theorem motivates the definition of square roots for positive semidefinite matrices.

Definition 33 *Let $M \in S_d(\mathbb{R})_0^+$. We define the* square root *of M as the unique matrix $N \in S_d(\mathbb{R})_0^+$ with $N^2 = M$. We denote it as $N = M^{1/2}$.*

We can also extend other concepts defined over the non-negative real numbers to the positive semidefinite matrices. For example, the square root allows us to define the concept of module for any matrix.

Definition 34 *Let $A \in \mathcal{M}_{d\times d'}(\mathbb{R})$. We define the* module *of A as*

$$|A| = (A^TA)^{1/2} \in S_{d'}(\mathbb{R})_0^+.$$

With the module we can state a polar decomposition theorem, which shows a decomposition that can be seen as an extension of the polar form for complex numbers.

Theorem 35 (Polar decomposition) *Let $A \in \mathcal{M}_{d\times d'}(\mathbb{R})$, with $d' \leq d$. Then, there is a matrix $U \in \mathcal{M}_{d\times d'}(\mathbb{R})$ with $U^TU = I$, so that $A = U|A|$. This decomposition is called the* polar decomposition *of A, and it is not necessarily unique, unless A is square and invertible.*

Proof First, observe that, given $x \in \mathbb{R}^{d'}$, we have

$$\|Ax\|_2^2 = (Ax)^T(Ax) = x^TA^TAx = x^T|A|^2x = x^T|A||A|x = (|A|x)^T(|A|x) = \||A|x\|_2^2.$$

This means that A and $|A|$ have the same effect on the length of any vector. As an immediate consequence, we can observe that $\ker A = \ker |A|$, since

$$x \in \ker A \iff Ax = 0 \iff \|Ax\| = 0 = \||A|x\| \iff |A|x = 0 \iff x \in \ker |A|.$$

As $d' = \dim\ker A + \dim\operatorname{im} A = \dim\ker|A| + \dim\operatorname{im}|A|$, we also conclude that $\dim\operatorname{im} A = \dim\operatorname{im}|A|$, and then $r(A) = r(|A|)$. We will denote this rank as $r \leq d$.

$|A|$ is positive semidefinite, so there is an orthonormal basis $\{w_1, \dots, w_{d'}\} \subset \mathbb{R}^{d'}$ consisting of eigenvectors of $|A|$, with corresponding eigenvalues $\lambda_1, \dots, \lambda_{d'}$. We can assume that $\lambda_1, \dots, \lambda_r > 0$ and $\lambda_{r+1} = \dots = \lambda_{d'} = 0$, or equivalently, $\{w_{r+1}, \dots, w_{d'}\}$ is an orthonormal basis of $\ker |A| = \ker A$.

We consider the set of vectors $\{Aw_1/\lambda_1, \dots, Aw_r/\lambda_r\}$. Note that

$$\left\langle \frac{1}{\lambda_i} Aw_i, \frac{1}{\lambda_j} Aw_j \right\rangle = \frac{1}{\lambda_i \lambda_j} \langle Aw_i, Aw_j \rangle = \frac{1}{\lambda_i \lambda_j} w_i^T |A|^2 w_j = \frac{1}{\lambda_i \lambda_j} w_i^T \lambda_j^2 w_j = \frac{\lambda_j}{\lambda_i} w_i^T w_j,$$

which equals 1 if $i = j$, and 0 otherwise, so this set is also orthonormal. In fact, this set is an orthonormal basis of $\operatorname{im} A$.

We extend the previous set to an orthonormal set of size d' in $\mathbb{R}^d$,

$$\left\{ \frac{1}{\lambda_1} Aw_1, \dots, \frac{1}{\lambda_r} Aw_r, v_{r+1}, \dots, v_{d'} \right\}.$$

Finally, we construct the matrix $V \in \mathcal{M}_{d \times d'}(\mathbb{R})$ by adding as columns the vectors in the previous set, and the matrix $W \in \mathcal{M}_{d'}(\mathbb{R})$ by adding as rows the vectors $w_1, \dots, w_{d'}$. We define U as $U = VW \in \mathcal{M}_{d \times d'}(\mathbb{R})$. Observe that both V and W have orthonormal columns, and then $V^T V = I = W^T W$, obtaining that $U^T U = I$ as well. We can also observe that $Ww_i = e_i$, where $\{e_1, \dots, e_{d'}\}$ is the canonical basis of $\mathbb{R}^{d'}$. Therefore, we obtain

$$Uw_i = \begin{cases} \frac{1}{\lambda_i} w_i, & 1 \le i \le r \\ v_i, & r < i \le d' \end{cases},$$

and finally,

$$U|A|w_i = \begin{cases} \lambda_i U w_i, & 1 \le i \le r \\ 0, & r < i \le d' \end{cases} = \begin{cases} Aw_i, & 1 \le i \le r \\ 0, & r < i \le d' \end{cases} = Aw_i,$$

where the last equality holds, since $\{w_{r+1}, \dots, w_{d'}\} \subset \ker A$. So, we have the equality $A = U|A|$ on the basis $\{w_1, \dots, w_{d'}\}$, concluding the proof. The uniqueness of U when A is square and invertible is due to the fact that $|A|$ is also invertible in that case, and then $U = A|A|^{-1}$.

■

Remark 36 *When $A \in \mathcal{M}_d(\mathbb{R})$ is a square matrix, the polar decomposition can be stated as $A = U|A|$, where $U \in O_d(\mathbb{R})$ is an orthogonal matrix.*

We are now in a position to prove Theorem 10.

Theorem *Let $M \in S_d(\mathbb{R})_0^+$. Then,*

1. *There is a matrix $L \in \mathcal{M}_d(\mathbb{R})$ so that $M = L^T L$.*
2. *If $K \in \mathcal{M}_d(\mathbb{R})$ is any other matrix with $M = K^T K$, then $K = UL$, where $U \in O_d(\mathbb{R})$ (that is, L is unique up to isometries).*

Proof The first statement was proved in Theorem 32. Suppose then that $L, K \in \mathcal{M}_d(\mathbb{R})$ verify that $M = L^TL = K^TK$. Let $L = V|L|, K = W|K|$, with $V, W \in O_d(\mathbb{R})$, be polar decompositions of L and K. Then, we have

$$\begin{aligned} L^TL = K^TK &\implies |L|^TV^TV|L| = |K|^TW^TW|K| \\ &\implies |L|^T|L| = |K|^T|K| \implies |L|^2 = |K|^2. \end{aligned}$$

As $|L|$ and $|K|$ are positive semidefinite, they must be the only square root of $|L|^2 = |K|^2$, that is, $|L| = |K|$. We call $N = |L| = |K|$. Returning to the polar decompositions of L and K, it follows that

$$N = V^TL = W^TK \implies K = WV^TL.$$

Therefore, taking $U = WV^T \in O_d(\mathbb{R})$, we obtain the desired equality.

■

Appendix B. Matrix Optimization Problems.

In this appendix we will develop a matrix optimization theory in order to prove Theorems 11, 12 and 13. First, we will introduce the Rayleigh quotient, and we will see its relationship with the eigenvalues and eigenvectors.

Definition 37 *Let $A \in S_d(\mathbb{R})$. We define the* Rayleigh quotient *associated with A as the mapping $\rho_A \colon \mathbb{R}^d \setminus \{0\} \to \mathbb{R}$ given by*

$$\rho_A(x) = \frac{x^TAx}{x^Tx} = \frac{\langle Ax, x\rangle}{\|x\|_2^2} \quad \forall x \in \mathbb{R}^d \setminus \{0\}.$$

If $B \in S_d(\mathbb{R})^+$, we define the generalized Rayleigh quotient *associated with A and B as the mapping $\mathcal{R}_{A,B} \colon \mathbb{R}^d \setminus \{0\} \to \mathbb{R}$ given by*

$$\mathcal{R}_{A,B}(x) = \frac{x^TAx}{x^TBx} = \frac{\langle Ax, x\rangle}{\|x\|_B^2} \quad \forall x \in \mathbb{R}^d \setminus \{0\}.$$

Throughout this section we will assume that $A \in S_d(\mathbb{R})$ and $B \in S_d(\mathbb{R})^+$ are fixed, and we will refer to Rayleigh quotients as $\rho = \rho_A$ and $\mathcal{R} = \mathcal{R}_{A,B}$. A first observation about ρ and $\mathcal{R}$ is that, for $x \in \mathbb{R}^d \setminus \{0\}$ and $\lambda \in \mathbb{R}^*$, it is verified that

$$\mathcal{R}(\lambda x) = \frac{(\lambda x)^TA(\lambda x)}{(\lambda x)^TB(\lambda x)} = \frac{\lambda^2(x^TAx)}{\lambda^2(x^TBx)} = \mathcal{R}(x).$$

Therefore, $\mathcal{R}$ takes all its values over the $(d-1)$-dimensional unit sphere, that is, $\mathcal{R}(\mathbb{R} \setminus \{0\}) = \mathcal{R}(\mathbb{S}^{d-1}) \subset \mathbb{R}$. Since $\mathcal{R}$ is continuous and the sphere is compact, it follows that $\mathcal{R}$ achieves a maximum and a minimum in $\mathbb{R}^d \setminus \{0\}$. The same follows with ρ. These maxima and minima are closely related with the problems we want to analyze. We start studying the extremes of ρ.

Theorem 38 (Rayleigh-Ritz) *Let $\lambda_{\min}$ and $\lambda_{\max}$ be the minimum and maximum eigenvalues of A, respectively. Then,*

1. *For every $x \in \mathbb{R}^d$, $\lambda_{\min}\|x\|^2 \leq x^TAx \leq \lambda_{\max}\|x\|^2$.*
2. *$\lambda_{\max} = \max_{x\in\mathbb{R}^d\setminus\{0\}} \frac{x^TAx}{x^Tx} = \max_{\|x\|_2=1} x^TAx$.*
3. *$\lambda_{\min} = \min_{x\in\mathbb{R}^d\setminus\{0\}} \frac{x^TAx}{x^Tx} = \min_{\|x\|_2=1} x^TAx$.*

Therefore, the maximum and minimum values of ρ are $\lambda_{\min}$ and $\lambda_{\max}$, respectively. These values are attained in the corresponding eigenvectors.

Proof Let $A = UDU^T$, with $U \in O_d(\mathbb{R})$ and $D = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$, where $\lambda_1 \leq \cdots \leq \lambda_d$, be a spectral decomposition of A. Let $x \in \mathbb{R}^d \setminus \{0\}$ and we take $y = U^Tx$. Then,

$$\rho(x) = \frac{x^TAx}{x^Tx} = \frac{x^TUDU^Tx}{x^Tx} = \frac{y^TU^TUDU^TUy}{y^TU^TUy} = \frac{y^TDy}{\|y\|_2^2} = \frac{\sum\limits_{i=1}^{d}\lambda_i y_i^2}{\|\lambda\|_2^2}. \tag{19}$$

In addition, it is clear that

$$\lambda_1\|y\|_2^2 = \lambda_1\sum_{i=1}^{d} y_i^2 \leq \sum_{i=1}^{d}\lambda_i y_i^2 \leq \lambda_d\sum_{i=1}^{d} y_i^2 = \lambda_d\|y\|_2^2.$$

Applying this inequality over Eq. 19, it follows that

$$\lambda_1 \leq \rho(x) \leq \lambda_d.$$

Furthermore, if u_1 and u_d are the corresponding eigenvectors of λ_1 and λ_d, we get

$$\rho(u_1) = \frac{u_1^TAu_1}{u_1^Tu_1} = \frac{\lambda_1u_1^Tu_1}{u_1^Tu_1} = \lambda_1, \quad \rho(u_d) = \frac{u_d^TAu_d}{u_d^Tu_d} = \frac{\lambda_du_d^Tu_d}{u_d^Tu_d} = \lambda_d.$$

Therefore, the equality is attained, and the three statements of the theorem follow from this equality. ■

Rayleigh-Ritz theorem shows us that $\rho(\mathbb{R}^d \setminus \{0\}) = [\lambda_{\min}, \lambda_{\max}]$, obtaining the extreme values in the corresponding eigenvectors. However, these are not the only eigenvalues that can act as an optimal for a Rayleigh quotient. If we restrict ourselves to lower dimensional spaces, we can obtain any eigenvalue of A as an optimal for the Rayleigh quotient, as we will see below.

Theorem 39 (Courant-Fischer) *Let $\lambda_1 \leq \cdots \leq \lambda_d$ the eigenvectors of A, and we denote by S_k a vector subspace of $\mathbb{R}^d$ of dimension k. Then, for each $k \in \{1, \dots, d\}$, we get*

$$\lambda_k = \min_{S_k\subset\mathbb{R}^d}\ \max_{\substack{x\in S_k \\ \|x\|_2=1}} x^TAx, \tag{20}$$

$$\lambda_k = \max_{S_{d-k+1}\subset\mathbb{R}^d}\ \min_{\substack{x\in S_{d-k+1} \\ \|x\|_2=1}} x^TAx. \tag{21}$$

This result extends the Rayleigh-Ritz statement, and this theorem is proven by Horn and Johnson (1990, chap. 4). There we can also find the proof of an important consequence of Courant-Fischer theorem, usually known as the Cauchy's interlace theorem.

Theorem 40 (Cauchy's interlace) *Suppose that $\lambda_1 \leq \dots \leq \lambda_d$ are the eigenvalues of A. Let $J \subset \{1, \dots, d\}$ be a set of cardinal $|J| = d'$, and let $A_J \in S_{d'}(\mathbb{R})$ be the matrix given by $A_J = (A_{ij})_{i,j \in J}$, that is, the submatrix of A with the entries of A whose indices are in $J \times J$. Then, if $\tau_1 \leq \dots \leq \tau_{d'}$ are the eigenvalues of A_J, for each $k \in \{1, \dots, d'\}$,*

$$\lambda_k \leq \tau_k \leq \lambda_{k+d-d'}.$$

The next result follows from Cauchy's interlace theorem, and the inequality it states will help us to solve our optimization problems.

Corollary 41 *Let $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$ with $LL^T = I$. If $\mu_1 \geq \dots \geq \mu_d$ are the eigenvalues of A and $\sigma_1 \geq \dots \geq \sigma_{d'}$ are the eigenvalues of LAL^T (now we are considering eigenvalues in decreasing order), then $\sigma_k \leq \mu_k$, for $k = 1, \dots, d'$.*

Proof Since $LL^T = I$, the rows of L are orthonormal eigenvectors. We can extend L to an orthogonal matrix $\hat{L} \in O_d(\mathbb{R})$ by adding $d - d'$ orthonormal eigenvectors, and orthonormal to the rows of L, in its rows. We have then that $\hat{L}A\hat{L}^T$ and A have the same eigenvalues, and LAL^T is a submatrix of $\hat{L}A\hat{L}^T$ obtained by deleting the last $d - d'$ rows and columns. The assertion now follows from Cauchy's interlace Theorem 40, considering eigenvalues in the opposite order. ■

We are now in a position to prove the theorems proposed in Section 2.2.3.

Theorem *Let $d', d \in \mathbb{N}$, with $d' \leq d$. Let $A \in \mathcal{S}_d(\mathbb{R})$, and we consider the optimization problem*

$$\begin{aligned} \max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad & \operatorname{tr}\left(LAL^T\right) \\ s.t.: \quad & LL^T = I. \end{aligned} \tag{22}$$

The problem attains a maximum if $L = \begin{pmatrix} — & v_1 & — \\ & \dots & \\ — & v_{d'} & — \end{pmatrix}$, where $v_1, \dots, v_{d'}$ are orthonormal eigenvectors of A corresponding to its d' largest eigenvalues. In addition, the maximum value is the sum of the d' largest eigenvalues of A.

Proof Let $\mu_1 \geq \dots \geq \mu_d$ the eigenvalues of A in decreasing order, and $\sigma_1 \geq \dots \geq \sigma_{d'}$ the eigenvalues of LAL^T. By Corollary 41, for any $L \in \mathcal{M}_{d' \times d}(\mathbb{R})$ with $LL^T = I$,

$$\operatorname{tr}(LAL^T) = \sum_{i=1}^{d'} \sigma_i \leq \sum_{i=1}^{d'} \mu_i.$$

In addition, when the rows of L are orthonormal eigenvectors $v_1, \dots, v_{d'}$ of A corresponding to $\mu_1, \dots, \mu_{d'}$, we get $LL^T = I$ and $\operatorname{tr}(LAL^T) = \sum_{i=1}^{d'} \mu_i$, thus equality holds for these vectors. ■

Lemma 42 (Simultaneous diagonalization) *Let $A \in S_d(\mathbb{R})$ and $B \in S_d(\mathbb{R})^+$. Then, there is an invertible matrix $P \in \mathrm{GL}_d(\mathbb{R})$ and a diagonal matrix $D \in \mathcal{M}_d(\mathbb{R})$ with $P^TAP = D$ and $P^TBP = I$.*

Proof We consider the matrix $C = B^{-1/2}AB^{-1/2}$. C is symmetric, since A and B are symmetric, thus there is a matrix $U \in O_d(\mathbb{R})$ so that U^TCU is diagonal. We call $D = U^TCU$ and we consider $P = B^{-1/2}U \in \mathrm{GL}_d(\mathbb{R})$. We get

$$P^TAP = P^TB^{1/2}CB^{1/2}P = (B^{-1/2}U)^TB^{1/2}CB^{1/2}(B^{-1/2}U) = U^TCU = D,$$
$$P^TBP = (B^{-1/2}U)^TB(B^{-1/2}U) = U^TB^{-1/2}BB^{-1/2}U = U^TU = I.$$

■

Theorem *Let $d', d \in \mathbb{N}$, with $d' \le d$. Let $A \in S_d(\mathbb{R})$ and $B \in S_d(\mathbb{R})^+$, and we consider the optimization problem*

$$\max_{L \in \mathcal{M}_{d' \times d}(\mathbb{R})} \quad \operatorname{tr}\left((LBL^T)^{-1}(LAL^T)\right) \tag{23}$$

The problem attains a maximum if $L = \begin{pmatrix} — & v_1 & — \\ & \dots & \\ — & v_{d'} & — \end{pmatrix}$, where $v_1, \dots, v_{d'}$ are eigenvectors of $B^{-1}A$ corresponding to its d' largest eigenvalues.

Proof We denote $U = L^T \in \mathcal{M}_{d \times d'}(\mathbb{R})$. If we take the matrix P from Lemma 42 and the matrix $V \in \mathcal{M}_{d \times d'}(\mathbb{R})$ with $U = PV$ (it exists and it is unique, since P is regular), we have

$$\begin{aligned} \operatorname{tr}\left((LBL^T)^{-1}(LAL^T)\right) &= \operatorname{tr}\left((U^TBU)^{-1}(U^TAU)\right) = \operatorname{tr}\left((V^TP^TBPV)^{-1}(V^TP^TAPV)\right) \\ &= \operatorname{tr}((V^TV)^{-1}(V^TDV)). \end{aligned}$$

Therefore, maximizing Eq. 23 is equivalent to maximize with respect to V the expression $\operatorname{tr}((V^TV)^{-1}(V^TDV))$, because the parameter change is bijective. Now we consider a polar decomposition $V = Q|V|$, with $Q \in \mathcal{M}_{d \times d'}(\mathbb{R})$ verifying $Q^TQ = I$. It follows that

$$\begin{aligned} \operatorname{tr}((V^TV)^{-1}(V^TDV)) &= \operatorname{tr}((|V|^TQ^TQ|V|)^{-1}(|V|^TQ^TDQ|V|^T)) \\ &= \operatorname{tr}(|V|^{-1}|V|^{-T}(|V|^TQ^TDQ|V|)) \\ &= \operatorname{tr}(|V|^{-1}Q^TDQ|V|) = \operatorname{tr}(Q^TDQ|V||V|^{-1}) = \operatorname{tr}(Q^TDQ). \end{aligned}$$

If we call $W = Q^T$, what we have obtained is that the maximization of Eq. 23 is equivalent to maximizing in W $\operatorname{tr}(WDW^T)$, subject to $WW^T = I$, thus obtaining the optimization problem given in Eq. 22. We can suppose the diagonal of D ordered in descending order, and then a matrix W that solves the optimization problem can be obtained adding as rows the vectors $e_1, \dots, e_{d'}$ of the canonical basis of $\mathbb{R}^d$. Then, Q has contains the same vectors, but added by columns. Observe that the quotient trace $T(X) = \operatorname{tr}\left((X^TBX)^{-1}(X^TAX)\right)$, with $X \in \mathcal{M}_{d\times d'}(\mathbb{R})$, is invariant with respect to right multiplications by invertible matrices. Indeed, if $R \in \mathrm{GL}_{d'}(\mathbb{R})$,

$$\begin{aligned} T(XR) &= tr\left((R^TX^TBXR)^{-1}(R^TX^TAXR)\right) = \operatorname{tr}(R^{-1}(X^TBX)^{-1}R^{-T}R^T(X^TAX)R) \\ &= \operatorname{tr}((X^TBX)^{-1}(X^TAX)RR^{-1}) = T(X). \end{aligned}$$

Since U maximizes T and $U = PQ|V|$, then PQ also maximizes T. In addition, as from $P^TAP = D$ and $P^TBP = I$ we obtain that

$$D = P^TAP = (P^TBP)^{-1}(P^TAP) = P^{-1}B^{-1}P^{-T}P^TAP = P^{-1}B^{-1}AP,$$

we conclude that P diagonalizes $B^{-1}A$, and then, it contains as columns the eigenvectors of this matrix. Since Q contains the d' first eigenvectors of the canonical basis by columns, PQ contains as columns the d' first eigenvectors of $B^{-1}A$, corresponding to its d' largest eigenvalues. This ends the proof, because a solution for the problem given by Eq. 23, which is equal to maximizing T except for a transposition, consists in adding those vectors as rows. ■

Theorem *Let $d', d \in \mathbb{N}$, with $d' \leq d$. Let $A, B \in S_d(\mathbb{R})^+$, and we consider the optimization problem*

$$\max_{L\in\mathcal{M}_{d'\times d}(\mathbb{R})} \quad \operatorname{tr}\left((LBL^T)^{-1}(LAL^T) + (LAL^T)^{-1}(LBL^T)\right) \tag{24}$$

The problem attains a maximum if $L = \begin{pmatrix} — & v_1 & — \\ & \dots & \\ — & v_{d'} & — \end{pmatrix}$*, where $v_1, \dots, v_{d'}$ are the d' eigenvectors of $B^{-1}A$ with the highest values for the expression $\lambda_i + 1/\lambda_i$, where λ_i is the eigenvalue associated with v_i.*

Proof First of all, given $C \in S_d(\mathbb{R})^+$ we consider the optimization problem

$$\max_{L\in\mathcal{M}_{d'\times d}(\mathbb{R})} \quad \operatorname{tr}\left((LCL^T + LC^{-1}L^T)\right) = \max_{L\in\mathcal{M}_{d'\times d}(\mathbb{R})} \quad \operatorname{tr}\left(L(C + C^{-1})L^T\right) \tag{25}$$

Using Theorem 22, a solution to this problem can be found by taking as rows of L the eigenvectors of $C + C^{-1}$ corresponding to its d' largest eigenvalues. Observe that the eigenvectors of C and C^{-1} are the same, and each one's eigenvalues are the inverse of the other. Therefore, $C + C^{-1}$ also has the same eigenvectors, and its eigenvalues have the form $\lambda + 1/\lambda$, for each λ eigenvalue of C. Then, the previous solution for Eq. 25 is equivalent to taking the eigenvectors of C for which $\lambda + 1/\lambda$ is maximized.

Finally, we only have to realize that we can follow the same proof as in Theorem 12, considering Eqs. 25 and 24 instead of Eqs. 22 and 23. ■

References

C. C. Aggarwal, Y. Zhao, and S. Y. Philip. On text clustering with side information. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 894–904. IEEE, 2012.

B. Ali, W. Kalintha, K. Moriyama, M. Numao, and K.-i. Fukui. Reinforcement learning for evolutionary distance metric learning systems improvement. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 155–156. ACM, 2018.

A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

A. Benavoli, G. Corani, F. Mangili, M. Zaffalon, and F. Ruggeri. A bayesian wilcoxon signed-rank test based on the dirichlet process. In *International conference on machine learning*, pages 1026–1034, 2014.

A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.

S. Boyd and J. Dattorro. Alternating projections. Technical report, Stanford University, 2003. Lecture notes of EE392o, Autumn Quarter.

S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217, 1967.

C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

T. Caliński and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

J. Carrasco, S. García, M. del Mar Rueda, and F. Herrera. rnpbst: An r package covering non-parametric and bayesian statistical tests. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 281–292. Springer, 2017.

D. Charte, F. Charte, S. García, and F. Herrera. A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations. *Progress in Artificial Intelligence*, in press, 2019.

R. Chatpatanasiri, T. Korsrilabutr, P. Tangchanachaianan, and B. Kijsirikul. A new kernelization framework for mahalanobis distance learning algorithms. *Neurocomputing*, 73 (10-12):1570–1579, 2010.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing).* Wiley-Interscience, 2006.

J. P. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *The Journal of Machine Learning Research*, 16(1):2859–2900, 2015.

B. Dacorogna. *Direct methods in the calculus of variations*, volume 78. Springer Science & Business Media, 2007.

J. V. Davis and I. S. Dhillon. Differential entropic clustering of multivariate gaussians. In *Advances in Neural Information Processing Systems*, pages 337–344, 2007.

J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

P. S. Dhillon, P. P. Talukdar, and K. Crammer. Inference-driven metric learning for graph construction. In *4th North East Student Colloquium on Artificial Intelligence*, 2010.

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in neural information processing systems*, pages 451–458, 2006.

J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2005.

M. Guillaumin, J. Verbeek, and C. Schmid. Is that you? metric learning approaches for face identification. In *Computer Vision, 2009 IEEE 12th international conference on*, pages 498–505, 2009.

Y. Hamamoto, S. Uchimura, and S. Tomita. A bootstrap technique for nearest neighbor classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19 (1):73–79, 1997.

I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan. The rise of big data on cloud computing: Review and open research issues. *Information systems*, 47:98–115, 2015.

N. J. Higham. Computing a nearest symmetric positive semidefinite matrix. *Linear algebra and its applications*, 103:103–118, 1988.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

R. A. Horn and C. R. Johnson. *Matrix analysis.* Cambridge university press, 1990.

I. Jolliffe. *Principal Component Analysis.* Springer Series in Statistics. Springer, 2002.

W. Kalintha, S. Ono, M. Numao, and K.-i. Fukui. Kernelized evolutionary distance metric learning for semi-supervised clustering. In *AAAI*, pages 4945–4946, 2017.

E. Kokiopoulou, J. Chen, and Y. Saad. Trace optimization and eigenproblems in dimension reduction methods. *Numerical Linear Algebra with Applications*, 18(3):565–602, 2011.

B. Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

Y. Lin and Y. Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

W. Liu, D. Xu, I. Tsang, and W. Zhang. Metric learning for multi-output tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Z. Ma and S. Chen. Multi-dimensional classification via a metric approach. *Neurocomputing*, 275:1121–1131, 2018.

X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *Computer Vision–ECCV 2012*, pages 488–501. Springer, 2012.

S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX, 1999. Proceedings of the 1999 IEEE signal processing society workshop.*, pages 41–48. Ieee, 1999.

P. Moutafis, M. Leng, and I. A. Kakadiaris. An overview and empirical comparison of distance metric learning methods. *IEEE transactions on cybernetics*, 47(3):612–625, 2017.

Y. Mu, W. Ding, and D. Tao. Local discriminative distance metrics ensemble learning. *Pattern Recognition*, 46(8):2337–2349, 2013.

B. Nguyen, C. Morell, and B. De Baets. Large-scale distance metric learning for k-nearest neighbors regression. *Neurocomputing*, 214:805–814, 2016.

B. Nguyen, C. Morell, and B. De Baets. Supervised distance metric learning through maximization of the jeffrey divergence. *Pattern Recognition*, 64:215–225, 2017.

B. Nguyen, C. Morell, and B. De Baets. Distance metric learning for ordinal classification based on triplet constraints. *Knowledge-Based Systems*, 142:17–28, 2018.

N. J. Nilsson. *Learning machines: foundations of trainable pattern-classifying systems*. McGraw-Hill, 1965.

M. L. Overton. On minimizing the maximum eigenvalue of a symmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 9(2):256–268, 1988.

N. Papernot and P. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

R. T. Rockafellar. *Convex analysis.* Princeton university press, 2015.

W. Rudin. *Real and complex analysis.* Tata McGraw-Hill Education, 1987.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

G. S. Sebestyen. *Decision-making processes in pattern recognition.* Macmillan, 1962.

S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

B. M. Steele. Exact bootstrap k-nearest neighbor learners. *Machine Learning*, 74(3):235–255, 2009.

M. Tan, I. W. Tsang, and L. Wang. Towards ultrahigh dimensional feature selection for big data. *The Journal of Machine Learning Research*, 15(1):1371–1429, 2014.

L. Torresani and K.-c. Lee. Large margin component analysis. In *Advances in neural information processing systems*, pages 1385–1392, 2007.

I. Triguero, S. González, J. M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M. J. del Jesús, L. Sánchez, and F. Herrera. Keel 3.0: an open source software for multi-stage analysis in data mining. *International Journal of Computational Intelligence Systems*, 10(1):1238–1249, 2017.

L. Van Der Maaten, E. Postma, and J. Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10:66–71, 2009.

F. Wang and C. Zhang. Feature extraction by maximizing the average neighborhood margin. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8, 2007.

L. Wang, B. Yang, Y. Chen, X. Zhang, and J. Orchard. Improving neural-network classifiers using nearest neighbor partitioning. *IEEE transactions on neural networks and learning systems*, 28(10):2255–2267, 2017.

K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

D. Wettschereck and T. G. Dietterich. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine learning*, 19(1):5–27, 1995.

X. Wu, X. Zhu, G.-Q. Wu, and W. Ding. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2014.

E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 521–528, 2003.

R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2(2):4, 2006.

X. Yang and Y. L. Tian. Eigenjoints-based action recognition using naive-bayes-nearest-neighbor. In *Computer vision and pattern recognition workshops (CVPRW), 2012 IEEE computer society conference on*, pages 14–19. IEEE, 2012.

Y. Ying and P. Li. Distance metric learning with eigenvalue optimization. *Journal of Machine Learning Research*, 13(Jan):1–26, 2012.

H. Zhang, A. C. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2126–2136. IEEE, 2006.

X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University, 2002.