Evolving linear transformations with a rotation-angles/scaling representation

A. Echeverría [a,*], J.M. Valls [b], R. Aler [b]

[a] Universidad Autónoma de Madrid, Madrid, Spain
[b] Universidad Carlos III de Madrid, Madrid, Spain

ARTICLE INFO

Keywords:
Data transformation
CMA-ES
Rotation angles
Scaling
Classification

ABSTRACT

Similarity between patterns is commonly used in many distance-based classification algorithms like KNN or RBF. Generalized Euclidean Distances (GED) can be optimized in order to improve the classification success rate in distance-based algorithms. This idea can be extended to any classification algorithm, because it can be shown that a GEDs is equivalent to a linear transformations of the dataset. In this paper, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is applied to the optimization of linear transformations represented as matrices. The method has been tested on several domains and results show that the classification success rate can be improved for some of them. However, in some domains, diagonal matrices get higher accuracies than full square ones. In order to solve this problem, we propose in the second part of the paper to represent linear transformations by means of rotation angles and scaling factors, based on the Singular Value Decomposition theorem (SVD). This new representation solves the problems found in the former part.

1. Introduction

Many classification algorithms use the concept of distance or similarity between patterns. This is specially true for local classification methods such as Radial Basis Neural Networks (Moody & Darken, 1989) or Nearest Neighbor classifiers (Cover & Hart, 1967). Typically, the Euclidean distance is used but this is not necessarily the best option for all classification domains. Therefore, finding the most appropriate distance for a classification task may be an important part in obtaining high classification rates. The Generalized Euclidean Distance (GED), displayed in Eq. (1), is a family of distances that depends on a parameter S, where S is a symmetric positive definite square matrix. This distance is usually called Mahalanobis distance (Atkenson, Moore, & Schaal, 1997; Tou & Gonzalez, 1974; Weisberg, 1985) although originally this term applied to cases where S is the inverse of the covariance matrix of the data. Eq. (1) shows the GED d_{ij} between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$.

$$d_{ij} = [(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{S} (\mathbf{x}_i - \mathbf{x}_j)]^{1/2} \quad (1)$$

In Valls, Aler, and Fernández (2007), Genetic Algorithms have been used to optimize GEDs distance matrices for Radial Basis Neural Networks. However, this approach can only be used in classification algorithms that explicitly use distances, like RBNN, but not others like C4.5. In order to make the approach more general, and by taking into account that a symmetric positive definite matrix S can always be decomposed into M^TM, for some matrix M, the definition of a GED results in Eq. (2).

$$d_{ij} = [(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M}^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)]^{1/2} \quad (2)$$

It can be shown that Eq. (2) is equivalent to computing an Euclidean distance on a transformed dataset, where the new patterns in the new space are linear transformations of the original data: $\mathbf{x}' = \mathbf{M}\mathbf{x}$. Thus, the GED can be redefined as shown in Eq. (3).

$$d_{ij} = [(\mathbf{M}\mathbf{x}_i - \mathbf{M}\mathbf{x}_j)^T (\mathbf{M}\mathbf{x}_i - \mathbf{M}\mathbf{x}_j)]^{1/2} = \left[(\mathbf{x}'_i - \mathbf{x}'_j)^T \left(\mathbf{x}'_i - \mathbf{x}'_j\right)\right]^{1/2} \quad (3)$$

Table 1 displays several examples of such transformations: (a) rotation of data points by θ degrees, (b) scaling of dimensions, and (c) projection onto the y coordinate.

The main goal of this paper is to use an optimization method, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), to look for a transformation matrix $\mathbf{M}$ as defined in Eq. (3) (instead of a GED) in order to improve the classification rate of a classification algorithm. CMA-ES is one of the best evolutionary techniques for continuous optimization in difficult non-linear domains (Hansen & Ostermeier, 2001; Ostermeier, Gawelczyk, & Hansen, 1994). CMA-ES is an Evolution Strategy where search is guided by a covariance matrix, which is adjusted and updated during the search process. CMA-ES works well in both local and global optimization

* Corresponding author.
E-mail addresses: alejandro.echeverria@uam.es (A. Echeverría), jvalls@inf.uc3m.es (J.M. Valls), aler@inf.uc3m.es (R. Aler).

Table 1
Examples of linear transformations: (a) rotation, (b) scaling, (c) projection.

(a) $M = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$

(b) $M = \begin{pmatrix} k & 0 \\ 0 & 1 \end{pmatrix}$

(c) $M = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$

tasks. One of the most interesting features of CMA-ES for our purposes is the self-adaptation of mutation (the update of the covariance matrix), allowing for a finely tuned local search at the end of the optimization process. In this paper, we will use CMA-ES to optimize the transformation matrix **M** for a classifier. Any learning algorithm could have been used but in the present work we will consider a nearest neighbor algorithm (KNN) with $K = 1$.

This paper reports the results of experiments using diagonal and square matrices M.[1] Experiments show that classification rates can be improved by both kind of matrices, but that in some cases, diagonal matrices outperform square matrices. This outcome is rather unexpected given that diagonal matrices are a subset of square ones. In order to deal with this issue, the second part of the paper tests a second representation of transformation matrices by using the Singular Value Decomposition theorem (SVD).

The SVD allows to decompose any real matrix into two rotation and one scaling matrices, as displayed in Eq. (4)

$$M = U\Sigma V^* \tag{4}$$

where V^* is the conjugate transpose of V. In our case, all the matrices have real values and then $V^* = V^T$. Thus,

$$M = U\Sigma V^T \tag{5}$$

The SVD is valid for any matrix, including rectangular ones. In this paper, matrices M are always square. In that case, U and V are both $n \times n$ unitary matrices and Σ is an $n \times n$ diagonal matrix with nonnegative real numbers on the diagonal. Both U and V^T are rotation matrices, that is, they transform data by performing a rotation in n-dimensions. This means that matrix M works by first rotating the data (V^T), then scaling it (Σ), and then rotating it again (U). However, let us remember that in this paper, we are using a nearest neighbor algorithm, which is rotation-invariant. This means that the final rotation U will not affect the classification results (this is also true of many other classification algorithms like Fisher Discriminant, Neural Networks, but not of others like C4.5). Therefore, in the current work and without losing generality, we will consider matrices of the form ΣV^T (i.e. a rotation followed by a scaling). Thus, our second method will evolve matrices Σ and V^T separately, instead of letting CMA-ES evolve matrix M. Moreover, the rotation matrix V^T can be encoded very naturally, by using explicitly the rotation angles. For instance, transformation (a) in Table 1 is a rotation θ in two dimensions represented in matrix form. But this transformation can be represented more naturally by encoding just the rotation angle θ.

We expected that this natural representation of linear transformations will not fall into the same pitfalls than the square matrices mentioned before: diagonal matrices outperformed square ones. The rationale for this is that SVD allows to factor the diagonal part (which is basically a scaling of the data, represented by matrix Σ) and the rotation part. By allowing the algorithm to separate both components, it is expected that CMA-ES will be able to find the proper scaling when only a scaling is needed. Experimental results will show that this is the case.

The structure of this article is as follows. Section 2 describes the CMA-ES algorithm. Section 3 describes the proposed methods, both evolving directly the transformation matrices and evolving the rotation angles and the scaling factors. Section 4 describes the synthetic and real domains that have been used to test the approach, and also reports the results of the experiments. Finally, Section 5 draws some conclusions and points to future work.

2. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

The algorithm CMA-ES (Covariance Matrix Adaptation Evolution Strategy) was proposed by Hansen (2009) and Hansen and Ostermeier (2001). It is an evolution strategy for difficult optimization problems in continuous domains, characterized by the use of a covariance matrix to guide the search process. In this section, we will give a very short overview of a (μ,λ) CMA-ES, where μ is the number of parents and λ the number of offspring. The full algorithm is quite complex, so the interested reader should consult the CMA tutorial for a complete description (Hansen, 2009).

CMA-ES estimates a probability distribution from the best performing samples in order to guide the search towards promising regions of the search space. Let us suppose that we desire to optimize a fitness function $f(x) : \mathbf{R}^p \rightarrow \mathbf{R}$, where p is the dimensionality of the problem. CMA-ES basic algorithm follows a randomized black box search, described as follows:

1. Initialize distribution parameters θ.
2. For generation (iteration) $t = 0,1,2,\ldots$:
   (a) Sample λ points from distribution $P(x|\theta^{(t)}) = x_1,\ldots,x_\lambda$.
   (b) Evaluate fitness f of $x_1,\ldots,x_\lambda$.
   (c) Update distribution parameters θ based on the best performers $x_1,\ldots,x_\mu$ ($f(x_1) \leqslant f(x_2) \leqslant \cdots f(x_\mu)$, for a minimization problem).

In CMA-ES, the probability distribution to be estimated is a multivariate normal $N(m,\delta^2 C)$, whose parameters θ are the mean m and the covariance matrix $\delta^2 C$. The mean represents the current location of the search and it moves towards better locations as search progresses. The covariance matrix controls mutations and is used to guide the search. In some sense, the covariance matrix "points" towards better solutions and is estimated from past samples x_i that performed well with respect to f. It is important to remark that CMA-ES decomposes the total covariance matrix into a covariance matrix C and the overall variance δ^2, also called the step-size control. The designers of the algorithm have found it useful to control the size of the mutation steps δ^2 independently of the direction of the search C. The reason is that estimating C requires many samples, and therefore takes longer to be adapted, whereas δ^2 needs to be adapted within shorter time-frames.

It has to be clarified that the mean, the step-size, and the covariance matrix are updated every generation t, so they should be written as $m^{(t)}$, $\delta^{(t)}$, $C^{(t)}$. However, in order to clarify the notation, the generation t will be omitted, by understanding that if the parameter appears on the right side of an equation, it belongs to generation t and to generation $t+1$ if it appears on the left side.

Sampling λ points from $N(m,\delta^2 C)$ can be easily done, according to Eq. (6) (where I is the identity matrix and $z \in \mathbf{R}^{\mathbf{p}}$ is a random vector generated from $N(0,I)$, an spherical multivariate normal distribution)

[1] In this paper, the term 'square matrix' will refer to matrices where all its components can be non-zero, in opposition to diagonal matrices, where only the diagonal components are non-zero.

$$x_i \leftarrow m + \delta \cdot \sqrt{C} \cdot z \tag{6}$$

After sampling, the best μ samples x_i are used to update the mean (m), the covariance matrix (C), and the step-size (δ). The update of the mean is straightforward, by means of Eq. (7), where w_i are user selected weights

$$m \leftarrow \sum_{i=1}^{\mu} w_i x_i \tag{7}$$

The update of C is done according to Eq. (8)

$$C \leftarrow (1 - c_{cov}) \cdot C + \alpha_1 \mathbf{Cov}(s_c) + \alpha_2 \mathbf{Cov}(x_{i=1\cdots\mu}) \tag{8}$$

where s_c is called the evolution path, and summarizes the best x_i samples from the beginning of the run (not just the ones in the current generation). In some sense, it contains a summary of the global performance so far. $\mathbf{Cov}(s_c)$ is the covariance due to the evolution path and $\mathbf{Cov}(x_{i=1\cdots\mu})$ is the covariance due to the best performers in the current generation. $c_{cov} \in (0,1]$ and $\alpha_1 + \alpha_2 = c_{cov}$ are learning ratios.

And finally, with respect to the adaptation of the step-size, δ may be either too long (and therefore it should be shortened) or too short (then it should be made larger). CMA-ES updates δ by taking into account the samples x_i along the evolution path. When the x_i in the path are correlated – all x_i are going in the same direction – the same distance could be covered in fewer but longer steps. Therefore, δ should be increased. On the other hand, when the evolution path is anticorrelated – the x_i go in contrary directions and cancel each other- the step-size δ should be made smaller so that region of the search space can be explored with a finer grain. In short, it is desirable that there are no correlations among samples along an evolution path. This can be done by comparing the length of the evolution path with the length of an evolution path under random selection, because random selection produces uncorrelated samples. Thus, the step-size is updated by means of Eq. (9)

$$\delta \leftarrow \delta \cdot \exp\left\{\beta \cdot \left(\frac{\|s_\delta\|}{\|E[N(0,I)]\|} - 1\right)\right\} \tag{9}$$

where β is a parameter that determines the step size variation between successive generations and, $\|E[N(0,I)]\|$ is the expectation of the length of a $N(0,I)$ distributed random vector in $\mathbf{R}^p$. Thus, Eq. (9) updates δ by comparing the length of the current evolutionary path $\|s_\delta\|$ and the expected length of an evolutionary path under random selection $\|E[N(0,I)]\|$.

3. The method

In this paper we have applied CMA-ES for finding data transformations that minimize the classification error (i.e. maximize the classification rate) of a learning technique. Two different ways of representing transformations have been tested. In the first method, transformations matrices coefficients are directly coded. In the second method, transformations are coded as rotation matrices and scalings. CMA-ES is extensively described in Ostermeier et al. (1994) and Hansen and Ostermeier (2001).[2]

3.1. First method: square and diagonal matrices

In the first method, we have considered two types of matrices: diagonal matrices (where all elements outside the diagonal are zero) and arbitrary non-diagonal matrices ($n \times n$ square matrices). Using diagonal matrices amounts to just a weighting of the attributes by the corresponding element in the diagonal, as shown in Eq. (10)

$$\begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix} * \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} k_1 * x_1 \\ k_2 * x_2 \end{pmatrix} \tag{10}$$

The reason for testing these two types of structures is to check whether square matrices are actually useful beyond a mere attribute weighting. Square matrices involve fitting many parameters ($n \times n$) and it is important to know whether they improve accuracy or rather produce overfitting because of the extra degrees of freedom, or underfitting because of not being able to adjust properly all the parameters.

In order to describe the application of any evolutionary algorithm, three elements have to be defined: the parameters, the chromosome, and the fitness function.

With regard to parameter setting, CMA-ES is almost parameter-free in the sense that it self-adjusts some of its own parameters during the search and also provide some reasonable values for the rest of parameters that usually work well. Typically, CMA-ES starts from a random individual, but in order to reduce some randomness in results, CMA-ES will start from the identity matrix for all the runs. This is equivalent to starting with no transformation (or starting with the Euclidean distance). Also, the initial mutation step size has been fixed to 1.0 (although CMA-ES will adjust this value in the course of the search). Also, a stopping criterion has to be provided. In this paper, we have checked empirically that 3000 fitness computations are enough for all domains.

With respect to the chromosome, CMA-ES individuals are matrices that represent linear transformations. Matrices can be directly encoded by flattening them (i.e. converting it into an array). If the matrix is square, the chromosome has n^2 components and is represented in Eq. (11)

$$\mathbf{v} = (x_{11}, x_{12}, \ldots, x_{21}, x_{22}, \ldots, x_{nn}) \tag{11}$$

If the matrix is diagonal, the chromosome is represented in Eq. (12), and has n components

$$\mathbf{v} = (x_{11}, x_{22}, x_{33}, \ldots, x_{nn}) \tag{12}$$

Finally, in order to evaluate each individual fitness, the original training dataset is transformed by the matrix encoded in the individual:

$$T_M = M * T \tag{13}$$

where M is the transformation matrix corresponding to the individual, T is the original training set and T_M is the transformed training dataset. Then, a classifier is trained using T_M. Any rotation-invariant learning algorithm L could have been used. In this paper, neighborhood-based algorithm KNN (with $k = 1$) has been selected. It is easy to think what kind of linear transformations could be useful for KNN, and this has helped in the design of some of the synthetic domains that will be described in the experimental section (Section 4). Fitness is then computed by performing a fivefold crossvalidation on T_M (Eq. (14)). Crossvalidation is required here because the training error of KNN is always zero

$$E_M = error(L, T_M) \tag{14}$$

3.2. Second method: rotation angles and scaling matrix

In the second method, a representation of rotation angles and scaling factors is directly used, to check whether the poor results produced in some domains with the square matrices of the first method, can be improved. Let us remember that in this second method, transformation matrices M will be represented by means of a scaling (diagonal) matrix Σ and a rotation matrix V (Eq. (15)). The later will be represented directly by rotation angles

$$M = \Sigma V^T \tag{15}$$

[2] We have used the Matlab code available at http://www.lri.fr/~hansen/cmaes_inmatlab.html.

The diagonal of the scaling matrix Σ will be represented as vector $\mathbf{s} = (s_1, \ldots, s_n)$, where n is the dimension of the problem. Matrix V will be represented as a vector of rotation angles $\mathbf{r} = (r_1, \ldots, r_a)$, where $a = n(n-1)/2$, are coded in the chromosome $\mathbf{v}$ as follows:

$$\mathbf{v} = (\mathbf{r}, \mathbf{s}) = (r_1, \ldots, r_a, s_1, \ldots, s_n),$$

where the first a elements are the rotation angles and the last n are the scaling factors. It means that our algorithm will perform $n(n-1)/2$ rotations and n scalings on data. Algorithmically, the generation of a n-dimensional rotation can be performed by multiplication of a rotation matrices $R(\alpha)$. In two dimensions, the form of these matrices that rotate a given vector by a counterclockwise angle is:

$$R(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{16}$$

In three dimensions, rotations of the three axes in a counterclockwise direction give the matrices in Eqs. (17)–(19), respectively (Arfken, 1985; Goldstein, 1980)

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \tag{17}$$

$$R_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix} \tag{18}$$

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{19}$$

The composition of rotation angles matrices $R_x(\alpha)R_y(\beta)R_z(\gamma)$ in three dimensions would give the transformation matrix of Eq. (20):

$$\begin{pmatrix} \cos\beta\cos\gamma & \cos\beta\sin\gamma & \sin\beta \\ -\cos\alpha\sin\gamma - \cos\gamma\sin\alpha\sin\beta & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & \cos\beta\sin\alpha \\ \sin\alpha\sin\gamma - \cos\alpha\cos\gamma\sin\beta & -\cos\gamma\sin\alpha - \cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\gamma \end{pmatrix} \tag{20}$$

As we can see in Bck (1996), only a few lines of code (Schwefel, 1980) are sufficient to perform these calculations. To perform our rotation-scaling transformation routine, we have added a scale transformation to this code producing Algorithm 1, explained below. First, the main loop is entered for all data instances p initializing a to the number of rotation angles, i.e., $a = n(n-1)/2$, where n is the domain dimension. Then, next two loops (lines 3 and 6) move data T indexes to perform 2D rotation obtaining a rotated data T_R (lines 8 and 9) in a counterclockwise direction, using a angles α contained in vector $\mathbf{r}$ (line 7). At the end of each iteration for every data instance, we perform a scale transformation multiplying it by vector $\mathbf{s}$ producing a final transformed data T_M (line 14).

Table 2
UCI domains characteristics.

Domain	Instances	Attributes	Classes
Blood (BL)	748	5	2
Car (CR)	1728	6	4
Diabetes (DB)	768	8	2
Ionosphere (IO)	351	34	2
Iris (IR)	150	4	3
Wine (WI)	178	13	3

Algorithm 1. Rotation Angles Routine

```
1: for i = 1 to p do
2:   a = n(n − 1)/2
3:   for j = 1 to n − 1 do
4:     n1 = n − j
5:     n2 = n
6:     for k = 1 to j do
7:       α = r(a)
8:       T_R(i,n2) = T(i,n1) sin(α) + T(i,n2) cos(α)
9:       T_R(i,n1) = T(i,n1) cos(α) − T(i,n2) sin(α)
10:      n2 = n2 − 1
11:      a = a − 1
12:    end for
13:  end for
14:  T_M(i) = T_R(i)s'
15: end for
```

The parameters setting for CMA-ES in the second method is similar to the first one: it starts with all the rotation angles set to zero and the scaling factors set to 1 (which corresponds to the identity transformation matrix). As in the first method, the initial step-size is set to 1.0 and the number of matrix evaluations allowed is the same than in the first method for each domain.

For the same reasons explained in Section 3.1 and with the aim of comparing both methods, KNN (with $k = 1$) is also used as classification algorithm.

4. Experiments

In this section, we will carry out experiments on several artificial and real world domains.

4.1. Domains

We have used four synthetic domains and six real world domains. All of them correspond to classification problems and have numerical attributes. They are described next.

4.1.1. Artificial data domains

We have used a well-known synthetic dataset, the **Ripley** (Ripley, 1996) data domain, which has been widely used in the Machine Learning literature, and three more artificial domains that we have called **RandomAttr**, **Straight-0** and **Straight-45**, specifically designed to check if the approach works properly.

In the **Ripley** (RP) dataset each pattern has two real-valued coordinates and a class which can be 0 or 1. Each class corresponds to a bimodal distribution that is a balanced composition of two normal distributions. Covariance matrices are identical for all the distributions and the centers are different. One of the issues that make this domain interesting is the big overlap existing between both classes.

RandomAttr (RA) is a two-class domain with four real-valued attributes x_1, x_2, x_3, x_4. The examples have been randomly generated following an uniform distribution in the interval [0,1] for attributes x_1, x_2 and the interval [0,100] for attributes x_3, x_4. If $x_1 < x_2$ then the example is labeled as class '1'. Otherwise, if $x_1 > x_2$ the example belongs to class '0'. Thus, attributes x_3 and x_4 are irrelevant. Because the ranges of irrelevant attributes are much bigger, the classification accuracy of KNN is very bad (about 50%). The dataset is composed of 300 examples, 150 from each class.

Straight-45 (S45) is a two-class domain with two real-valued attributes. The examples have been generated in this way: initially 100 examples of class 1 are located as regular intervals in a straight

line passing through the origin (0,0) with an angle of 45° respect to the horizontal axe. The distance between two consecutive points is 1. One hundred examples of class 0 are generated in the same way in a parallel straight line passing through the point $(0,-1)$ in such a way that the nearest point of a given point always belongs to the opposite class, because it is located in the opposite parallel straight line. Then all points are perturbed by adding to each coordinate a random number uniformly distributed in $[-0.5, 0.5]$.

The idea behind this domain is that the nearest neighbor algorithm will achieve a very bad result because most of the times the nearest neighbor of a given point belongs to the opposite class. But certain transformations of the data involving rotations and coordinate scaling will allow a good classification rate.

Straight-0 (S0) is very similar to **Straight-45**. The only difference is that all the points have been rotated 45° clockwise. The motivation for using this domain is that in this case with a simpler transformation the data could be properly classified because no rotation is needed. In short, **Straight-45** requires both rotation and scaling (a square matrix) whereas **Straight-0** requires scaling only (a diagonal matrix).

4.1.2. Real world data domains

We have used the well known **Blood** (BL), **Car** (CR), **Diabetes** (DB), **Ionosphere** (IO), **Iris** (IR) and **Wine** (WI) domains from the UCI Machine Learning Repository.[3] Table 2 displays the characteristics of these UCI domains.

4.2. Experimental results. Square vs. diagonal matrices

In this subsection we show the experimental results obtained when square and diagonal matrix transformations are evolved by CMA-ES and KNN (with $k = 1$) is used as classifier in the transformed space. The parameters for CMA-ES are the following: the initial standard deviation has been set to 1.0 for all the experiments. The maximum number of fitness evaluations has been set to the same value (3000) for both methods by means of preliminary experiments. The rest of CMA-ES parameters are set to their default values.

In all domains we compare the classification results in three situations: for each fold, KNN classifies the original data, the data transformed by a diagonal matrix optimized by CMA-ES and by a square matrix optimized by CMA-ES. In all cases linear transformations are done by means of complete matrices and thus, the dimension of the transformed space remains unaltered.

Table 3 shows the mean classification accuracy rates obtained for all the domains and their standard deviations. The means have been obtained by averaging 10 executions of a 10-fold crossvalidation procedure. The best results have been marked with a dagger. A Corrected Repeated 10 × 10-fold Crossvalidation T-Test (Bouckaert & Frank, 2004) has been used in all cases for testing if differences between an algorithm and the results obtained with the original data are statistically significant ($\alpha = 0.05$). Statistically significant results are marked with a dagger (†). The last row of Table 3 counts in how many domains the differences are significant.

In the **Straight-0** domain, with a simple transformation of the space, just compressing the x_1 coordinate, a good classification rate can be achieved because points belonging to the same class can get as close to each other as needed. This simple transformation can be done with a diagonal matrix and therefore with a square matrix too. The results show that KNN only obtains a classification rate of 9.2% on the original dataset, but if the data is linearly transformed with a diagonal matrix the rate is 99.7% and 99.8% if a square matrix is used. In this case both optimization methods achieve the same results for each matrix type.

[3] http://archive.ics.uci.edu/ml/.

Table 3
Classification rate (percentage) with KNN ($k = 1$) for the original dataset and for the transformed data when square and diagonal matrix transformations are used.

	Original data	CMA-ES diagonal	CMA-ES square
Straight-0 (S0)	9.20 ± 2.26	†99.70 ± 0.79	†99.85 ± 0.24
Straight-45 (S45)	9.20 ± 1.62	8.95 ± 1.89	†99.15 ± 0.58
RandomAtt (RA)	50.50 ± 2.38	†95.60 ± 1.81	†79.63 ± 6.28
Ripley (RP)	88.58 ± 0.36	88.91 ± 0.44	88.41 ± 0.57
Blood (BL)	68.41 ± 0.57	67.92 ± 0.69	68.12 ± 1.01
Car (CR)	87.49 ± 0.17	†95.82 ± 0.33	†97.52 ± 0.18
Diabetes (DB)	68.11 ± 1.02	67.10 ± 1.51	67.28 ± 0.95
Ionosphere (IO)	86.19 ± 0.39	89.94 ± 1.57	88.31 ± 0.63
Iris (IR)	96.00 ± 0.00	94.80 ± 0.76	95.27 ± 1.24
Wine (WI)	75.99 ± 1.47	†94.39 ± 1.14	†85.69 ± 1.64
Significantly better/worse		4/0 (of 10)	5/0 (of 10)

In the **Straight-45** domain, we see that a more complex transformation must be done because it is not enough to scale the coordinates but to rotate the points as well. This linear transformation cannot be obtained with a diagonal matrix. The second row of Table 3 shows the results as expected: KNN obtains a very bad classification accuracy on the original data (9.20%). A diagonal matrix is useless to obtain an adequate transformation and we can see that the results are very bad too, around 8.95%. On the contrary, when a square matrix is used, the results are near to 100%.

The **RandomAttr** domain has two irrelevant attributes whose numeric range is much bigger that the relevant attributes range. That is the reason why the accuracy of KNN on the original data is 50.5%. Scaling the irrelevant attributes should be enough to attain a much better accuracy, thus both diagonal and square matrices should be appropriate. The results show that diagonal matrices obtain results over 90%. It can also be seen that in this domain, square matrices do not perform as well as diagonal matrices.

With regard to the remaining domains, CMA-ES diagonal obtains significatively better results than KNN on the original data for Car (95.82% vs. 87.49%), and Wine (94.39% vs. 75.99%). CMA-ES square also obtains significatively better results than KNN on the original data for the same domains: Car, 97.52% vs. 87.49% and Wine, 85.69% vs. 75.99%.

Summarizing the results, it can be observed that CMA-ES diagonal and CMA-ES square behave as expected in the artificial domains, except in RandomAttr, where CMA-ES square obtains much worse results than expected. Also, it is never the case that the original data significantly outperforms any of the other two methods. There are some cases where diagonal matrices largely outperform square ones (RandomAttr and Wine). In those two domains, the square matrix results are much worse than the diagonal ones. This is remarkable, given that the set of square matrices include diagonal ones. This issue deserves more attention because it would be useful to apply just one method to all domains: the evolution of square matrices.

Square matrices have more parameters to fit than diagonal ones, but the worse results of square matrices in RandomAttr and Wine is not a case of overfitting. We have extended the number of evaluations from 3000 to 10,000, and results in RandomAttr improve from 79.63% to 88.86%, and in Wine from 85.69% to 87.14%. These improvements hint that the issue here is not overfitting but rather underfitting due to a very slow convergence. In any case, RandomAttr is a very simple problem and requiring more than 10,000 evaluations in order to obtain the same results than a diagonal matrix does not seem appropriate. In order to deal with this issue, we have proposed a second way of representing the transformation matrix. This second method was described in Section 3.2. Next section (Section 4.3) reports the experimental results).

Table 4
This table adds to Table 3 a new column for our classification rate (percentage) with KNN ($k = 1$) for the transformed data when scaling factors and rotation angles are directly evolved.

	Original data	CMA-ES diagonal	CMA-ES square	CMA-ES rotation angles
Straight-0 (S0)	9.20 ± 2.26	†99.70 ± 0.79	†99.85 ± 0.24	†98.90 ± 0.66
Straight-45 (S45)	9.20 ± 1.62	8.95 ± 1.89	†99.15 ± 0.58	†98.60 ± 0.32
RandomAtt (RA)	50.50 ± 2.38	†95.60 ± 1.81	†79.63 ± 6.28	†97.70 ± 0.46
Ripley (RI)	88.58 ± 0.36	88.91 ± 0.44	88.41 ± 0.57	88.33 ± 0.68
Blood (BL)	68.41 ± 0.57	67.92 ± 0.69	68.12 ± 1.01	67.31 ± 0.89
Car (CR)	87.49 ± 0.17	†95.82 ± 0.33	†97.52 ± 0.18	†96.23 ± 0.37
Diabetes (DB)	68.11 ± 1.02	67.10 ± 1.51	67.28 ± 0.95	67.95 ± 1.07
Ionosphere (IO)	86.19 ± 0.39	89.94 ± 1.57	88.31 ± 0.63	†91.51 ± 0.98
Iris (IR)	96.00 ± 0.00	94.80 ± 0.76	95.27 ± 1.24	94.93 ± 0.95
Wine (WI)	75.99 ± 1.47	†94.39 ± 1.14	†85.69 ± 1.64	†96.37 ± 0.80
Significantly better/worse		4/0 (of 10)	5/0 (of 10)	6/0 (of 10)

4.3. Experimental results. Rotation angles and scaling

In this subsection we show the experimental results obtained by the second method described in Section 3.2 when scaling factors and rotation angles are evolved and KNN (with $k = 1$) is used as classifier in the transformed space. The parameters for CMA-ES are the same as in the previous subsection. We apply the method to the same data folds previously used when the first method was applied. Thus, the results can be compared with the ones displayed in Table 3 and obtained when the transformation matrices were directly evolved.

In order to allow a good comparative of both methods, Table 4 shows all the results: column 1 shows the classification rates when KNN is applied to the original data. Columns 2 and 3 show the results corresponding to the first method explained in Section 4.2, when either diagonal and square matrices were directly evolved. Finally, column 4 displays the results obtained by the second method, when scaling factors and rotation angles are evolved. In all cases the results correspond to the mean classification accuracy rates obtained for all the domains and their standard deviations. As in the last subsection, the means have been obtained by averaging 10 executions of a 10-fold crossvalidation procedure. The best results have been marked with a dagger. As before, a Corrected Repeated 10×10-fold Crossvalidation T-Test (Bouckaert & Frank, 2004) has been used for testing if each method is significantly better (or worse) than KNN on the original data ($\alpha = 0.05$). A dagger (†) marks significant differences.

In the **Straight-0** domain, as we said before, just compressing the x_1 coordinate, a good classification rate can be achieved because points belonging to the same class can get as close to each other as needed. This simple transformation can be done with a diagonal matrix, and therefore with a square matrix and a rotation-angles matrix too. The results show that KNN only obtains a classification rate of 9.2% on the original data set, but if the data is linearly transformed with a square matrix the rate is 99.8% and 98.9% if a rotation-angle matrix is used.

In the **Straight-45** domain, a rotation is needed. Table 3 shows the results as expected: KNN obtains a very bad classification accuracy on the original data (9.2%) and when a diagonal matrix is used (8.95%). On the contrary, when a square matrix or rotation-angles matrix is used, the results are near to 100%.

In the **RandomAttr** domain, scaling the irrelevant attributes should be enough to attain a much better accuracy. Therefore, diagonal matrices should be enough but given that square matrices include diagonal ones, square matrices should get similar results. But we seen in Table 3 that square matrices did not reach the expected performance. On the other hand, rotation-angles matrices perform very well, and they obtain a rate of 97.70% vs. 79.63% for square matrices. In this domain we observe that encoding directly the scaling factors and the rotation angles, facilitates the search of the appropriate transformation.

Regarding the real data domains, we can see that the rotation-angles method also solves the stagnation problem for the Wine domain: 96.37% (rotation angles) vs. 75.99% (square matrix) vs. 94.39% (diagonal matrix).

Summarizing the results, it can be observed that the CMA-ES rotation-angles method behaves as expected in all domains, avoiding the situations where the first method (evolving directly the square matrices) got stagnated in some domains (RandomAttr and Wine). Rotation-angles is also the method that obtains the highest number of statistically significant differences with the original data: 6 out of 10 vs. 5 out of 10 (square matrices) and 4 out of 10 (diagonal matrices).

5. Conclusions

In this work, the evolutionary method CMA-ES is used to search for appropriate linear transformations of data that improve the success rate of KNN. Working with linear transformations has an important advantage: any learning algorithm with numerical attributes can be used, even if it is not based on distances.

In a first stage, linear transformations have been represented as straightforward matrices. Both diagonal and full square matrices have been considered. The method has been tested with different domains, both synthetic and real, and the results show that, in general, both diagonal and square matrices found by CMA-ES either outperform or match the classifier on untransformed data. Square matrices have the advantage that they allow for more complex transformations that are required in some of the domains. However, in certain domains diagonal matrices attain significatively better results than square ones. This is unexpected given that diagonal matrices are a subset of square ones. We have tested that this is not due to overfitting but rather to a very slow convergence in some of the domains.

In order to deal with this issue, we have modified the method using a different representation of transformation matrices, which is based on the Singular Value Decomposition theorem (SVD). According to SVD, any linear transformation can be split into rotations and scalings. Thus, in the second part of this work, the transformations are directly represented by rotation angles and scaling factors. By allowing the algorithm to separate both components, it is expected that CMA-ES will be able to find the proper scaling when only a scaling (or diagonal matrix) is needed. Experimental results show that the rotation-angles representation manages to avoid the stagnation problem of square matrices.

In the future, we will test what other learning methods benefit from data transformations. Also, the rotation-angles method could

be used as a dimensionality reduction technique in a similar vein to Sierra and Echeverra (2006).

Acknowledgment

This article has been financed by the Spanish founded research MCINN project MSTAR::UC3M, Ref:TIN2008-06491-C04-03, and by project A3::UAM, Ref:TIN2007-66862-C02–02.

References

Arfken, G. (1985). *Mathematical methods for physicists* (third ed.). Orlando, FL: Academic Press.

Atkenson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review, 11*, 11–73.

Bck, T. (1996). *Evolutionary algorithms in theory and practice*. Oxford New York: Oxford University Press.

Bouckaert, R. R., & Frank, E. (2004). Evaluating the replicability of significance tests for comparing learning algorithms. *Advances in Knowledge Discovery and Data Mining (PAKDD)*, 3–12.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21–27.

Goldstein, H. (1980). *Classical mechanics* (second ed.). Reading, MA: Addison-Wesley.

Hansen, Nikolaus (2009). *The CMA evolution strategy: A tutorial*. TU Berlin: Technische Universitat Berlin.

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation, 9*(2), 159–195.

Moody, J. E., & Darken, C. (1989). Fast learning in networks of locally tuned processing units. *Neural Computation, 1*, 281–294.

Ostermeier, A., Gawelczyk, A., & Hansen, N. (1994). A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation, 4*(2), 369–380.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge: Cambridge University Press.

Schwefel, Hans-Paul (1980). Subroutines `EVOL`, `GRUP`, `KORR` – Listings and user's guides. *Nuclear Research Centre (KFA) Jülich*..

Sierra, A., & Echeverra, A. (2006). Evolutionary discriminant analysis. *IEEE Transactions on Evolutionary Computation, 10*(1), 81–92.

Tou, J. T., & Gonzalez, R. C. (1974). *Pattern recognition principles*. Addison-Wesley.

Valls, J. M., Aler, R., & Fernández, O. (2007). Evolving generalized euclidean distances for training RBNN. *Computing and Informatics, 26*, 33–43.

Weisberg, S. (1985). *Applied linear regression*. New York: John Wiley and Sons.